VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A REALIZACE KVADROKOPTÉRY S VYUŽITÍM ARDUINO DUE

QUADCOPTER DESIGN AND UTILIZATION WITH THE USE OF ARDUINO DUE

AUTOR PRÁCE
AUTHOR
Dominik Majer

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Ondřej Krajsa, Ph.D.

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Dominik Majer ***ID:*** 164761
Ročník: 3 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Návrh a realizace kvadrokoptéry s využitím Arduino Due

POKYNY PRO VYPRACOVÁNÍ:

S využitím kitu Arduino Due realizujte dálkově řízenou kvadrokoptéru. Do zařízení implementujte metody pro zabránění srážky s překážkou a lokalizaci. Analyzujte možnosti autonomního řízení podle předepsané trasy a možnosti telemetrie.

DOPORUČENÁ LITERATURA:

[1] WILCHER, Don. Arduino Electronics Blueprints. Packt Publishing, 2015. ISBN 9781784393601.

[2] OGATA, Katsuhiko. Modern control engineering. 5th ed. Boston: Prentice Hall, 2010, x, 894 s. ISBN 978-0--3-615673-4.

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 1.6.2016

Vedoucí práce: Ing. Ondřej Krajsa, Ph.D.
Konzultant bakalářské práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ABSTRAKT

Bakalářská práce se věnuje návrhu funkčního prototypu kvadrokoptéry za použití standardních modelářských komponent s řídící jednotkou Arduino Due a gyroskopem obsahující čip MPU6050. Cílem je funkční (letu-schopný) prototyp spolu s mechanismem redukce šance případné srážky s překážkou za použití ultrazvukových senzorů. V první části je proveden popis jednotlivých komponent a jejich součinnosti, ve druhé části pak popis konečného řešení spolu s popisem vývojového diagramu, zdrojového kódu a vzniklými problémy. Zdrojový kód je napsán v programovacím jazyku Arduino, který je založen na jazyku Wiring. Dále je provedena stručná analýza možností autonomního řízení kvadrokoptéry, konkrétně za pomoci GPS modulu a nastínění její realizace. Nakonec jsou zanalyzovány základní způsoby přenosu telemetrie jako jsou přenosy pomocí GSM modulu, wi-fi, bluetooth apod.

KLÍČOVÁ SLOVA

Kvadrokoptéra, Arduino, Arduino Due, RC, BLDC, ARM

ABSTRACT

This bachelor thesis is about design of usable prototype of quadrocopter using standard RC components, with Arduino Due control unit and gyroscope with MPU6050. Main goal is the fully functional prototype with feature for reduction of barriers collision possibility, using ultrasonic sensors. The first part of thesis is the description of each component and their cooperation. The second part describes final solution, flowchart, source code with problems arisen. Source code is written in programming language Arduino, which is based on Wiring language. Further the brief analysis of autonomous flight of quadrocopter is done, particulary with GPS module and possible brief design of this idea. Finally the analysis of basic options of telemetry transmission, such as GSM module, wi-fi, bluetooth, are analysed.

KEYWORDS

Quadrocopter, drone, quadrotor, Arduino, Arduino Due, RC, BLDC, ARM

MAJER, Dominik *Návrh a realizace kvadrokoptéry s využitím Arduino Due*: bakalářská práce. Místo: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 66 s. Vedoucí práce byl Ing. Ondřej Krajsa, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Návrh a realizace kvadrokoptéry s využitím Arduino Due“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Místo ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Ondřeji Krajsovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. A především bych rád poděkoval mým rodičům, kteří mě po celé tři roky studia bezvýhradně podporovali.

Místo ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Místo . . . . . .

. . . . . .
(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Kvadrokoptéry jsou v dnešní době velmi rozšířený fenomén. Zejména díky běžné dostupnosti modelářských dílů k tomu určených a též díky určité postupné normalizaci těchto dílů. Já se v této práci pokusím problematiku sestavení a zprovoznění zpracovat. Na rozdíl od běžných RC modelářů nevyužiji k řízení předprogramovanou řídící jednotku obsahující vlastní gyroskop, ale jako řídící jednotku jsem zvolil vývojovou desku z rodiny Arduin – Arduino Due. K tomu připojím gyroskop s čipem MPU6050, který bude s Arduinem Due komunikovat po sběrnici TWI, což je obdobou sběrnice I^2C. Takže když pominu fyzické sestavení, jsou hlavními úkoly, zjistit jakým způsobem lze Arduinem Due ovládat motory (respektive ESC regulátory), přijímat příkazy z vysílače, ale tedy hlavně napsat řídící kód k celkovému řízení. Dalším úkolem je zabudovat do kódu redukci šance srážky s překážkou za použití ultrazvukových senzorů. Samotné psaní kódu je prováděno ve vývojovém prostředí Arduino a v programovacím jazyku Arduino. Jako dva menší úkoly byly zadány analýza autonomního řízení a způsoby přenosu telemetrie. V analýze autonomního řízení se stručně podívám na řízení za použití GPS modulu. Tzn. autonomní let po zadání určitých GPS souřadnic. Nakonec rozeberu základní způsoby pro přenos telemetrie za použití GSM modulu, wi-fi, bluetooth apod.

1 TEORETICKÉ ŘEŠENÍ A POPIS POUŽITÝCH KOMPONENT

1.1 Kvadrokoptéra obecně

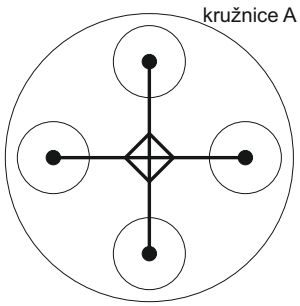
Kvadrokoptéra, někdy taky známá pod názvem *dron* či anglické ekvivalenty *quadrocopter*, *drone* a občas se též setkáme s výrazem *quadrotor*. První funkční stroj vzhledově odpovídající podobě dnešních kvadrokoptér, vyrobil Etienne Oehmichen roku 1920. První kvadrokoptéry byly stavěny za účelem aby nesly lidskou posádku, což se ale neprosadilo a v dnešní době je jejich vývoj soustředěn především na malé bezpilotní letouny. Jejich využití je v dnešní době velmi rozsáhlé. Začínaje od již hotových modelů, které jsou běžně ke koupi v obchodech se vším všudy (RC modely na ovládání), přes modelářské kvadrokoptéry, kdy si lze koupit všechny nezbytné komponenty a jednoduše vše složit, až po profesionální použití např. u stavebních firem, kdy se na těžce přístupné místa vyšle kamerou vybavená kvadrokoptéra za účelem např. revize venkovních stran výškových budov nebo hrází u vodních nádrží apod. V neposlední řadě našli široké uplatnění u policie, hasičů nebo armády, kdy můžou pomáhat např. při výškovém ohledání místa případné katastrofy/nehody. Určitým způsobem zajímavé je též použití drogovými pašeráky pro přenos drog přes hranice a tím neriskovat zatčení nebo možnou krádež/zabavení „vzácného“ nákladu při pozemní přepravě. Rozsáhlé je i nasazení ve filmovém průmyslu pro záběry z ptačí perspektivy. Nejvíce se v praxi můžeme setkat právě s kvadrokoptérami, ale poměrně rozšířené jsou např. i trikoptéry nebo hexakoptéry (jména dle počtu nosných vrtulí). U každého typu je ale možných několik odlišných způsobů řízení. Ilustrační obrázek komerčně prodávané kvadrokoptéry viz obr. 1.1.

Obr. 1.1: Ilustrační obrázek komerčně prodávané kvadrokoptéry, převzato z [1]

1.1.1 Teoretická realizace kvadrokoptéry

Na úvod k mému řešení musím říci, že svým zadáním se liší od standardních projektů, kdy buď např. jako amatérský modelář koupím všechny součástí a naprogramovanou řídící jednotku a jen všechno sestavím nebo projekty zaměřené spíše na určitou součást kvadrokoptéry (např. na stabilizaci, motory...). Já mám zadané pouze použití Arduina Due. Proto jsem se rozhodl na celou věc podívat sám za sebe a jít svojí vlastní cestou.

Nejprve jsem si zvolil přední stranu nebo obecně řečeno bod – označme si ho *výchozí bod* – od kterého se bude odvíjet smysl pohybu v prostoru, tzn. kam kvadrokoptéra poletí když řekneme *leť vpřed*. Ještě je třeba nejprve říct o osách *yaw*, *pitch* a *roll*. Toto označení se využívá v letectví (tudíž i pro kvadrokoptéry). V podstatě se jedná o analogické vyjádření pro osy x, y a z, tak jak jsme je zvyklí využívat pro popis trojrozměrného prostoru. Takže matematicky řečeno – vektor pohybu kvadrokoptéry bude mít směr přímky od průsečíku os *yaw* a *pitch* do výchozího bodu, viz obr. 1.2). Možností je ale nekonečně mnoho, jelikož to může být každý bod na kružnici A na obr.1.2 (všechno nyní uvažujeme ve dvourozměrném, vodorovném prostoru).



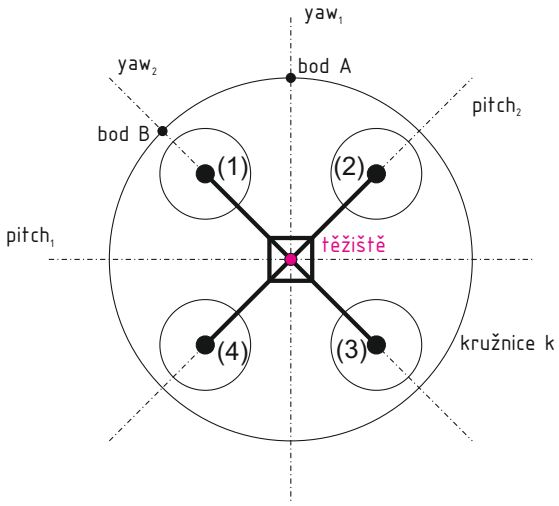
Obr. 1.2: Teoretické možnosti pohybu kvadrokoptéry

Z obr.1.2 tedy vyplývá, že je nekonečně mnoho možností kde umístit výchozí bod. Při popisu výchozího bodu se budeme pohybovat pouze ve dvourozměrném vodorovném prostoru, tzn. osu *roll* neuvažujeme. Takže podle obr. 1.3 je první ze dvou pro mě nejlépe vycházejících možností, umístění os yaw_1 a $pitch_1$, pro které platí jako výchozí bod, bod A. U tohoto řešení ovšem nastává problém při rozdílném výkonu jednotlivých motorů, protože pokud např. budeme chtít letět vpřed, tzn. těžištěm[1] směrem k výchozímu bodu (bod A), musíme zvýšit tah motorů (3) a (4) a o stejnou hodnotu snížit tah motorů (1) a (2). Nabízí se zde sice možnost pouze zvýšit tah motorů (3) a (4) (respektive pouze snížit tah motoru (1) a (2)), ale to by se nezachoval vodorovný let a kvadrokoptéra by zároveň při pohybu dopředu stoupala

[1] Těžiště značené na obrázcích a zmiňované v textu je myšleno pro ideální případ kvadrokoptéry, kdy je celá dokonale vyvážena.

(respektive klesala). S tímto nedostatkem si neporadí ani virtuální PID regulátory, jelikož musíme znát o kolik méně či více mají jednotlivé motory tah vůči nějaké výchozí hodnotě.

Druhou možností je umístění dle os yaw_2 a $pitch_2$ na obr. 1.3. U tohoto řešení je jako výchozí bod, bod B. Z řešení vyplývá, že pokud budeme chtít letět dopředu (směrem k bodu B), budeme muset zvýšit výkon pouze motoru (3) a ne nutně o stejnou hodnotu snížit výkon motoru (1). Zatímco s motory (2) a (4) se neděje nic a zůstávají tak, aby osa $pitch_2$ byla vodorovně (analogicky stejný princip platí při letu vzad, doprava a doleva). Tzn. pro každý motor nám stačí jeden virtuální PID regulátor a částečně se nám vyřeší problém s rozdílným tahem motorů.



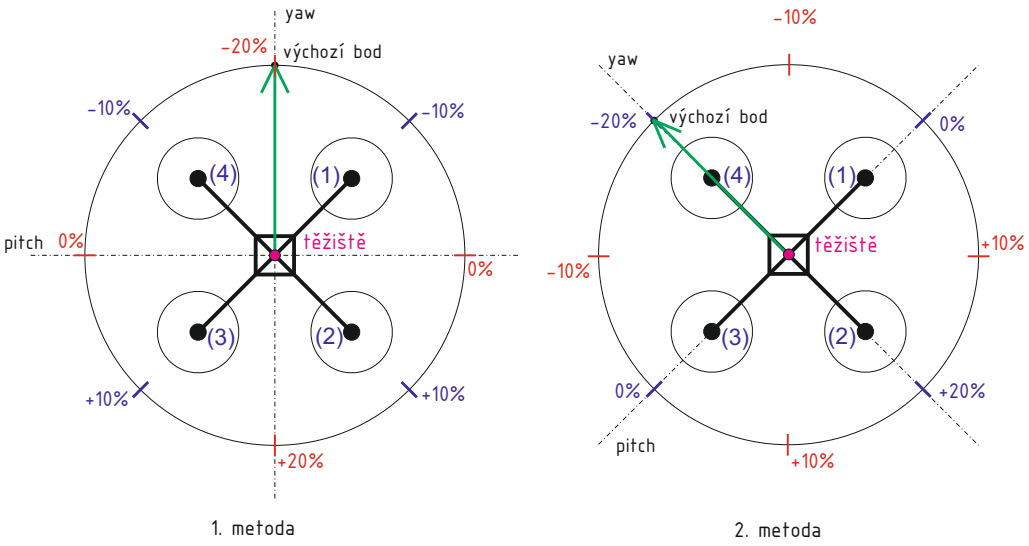
Obr. 1.3: Rozvržení os *yaw* a *pitch* a výchozího bodu s označením motorů

Ovšem roli zde hraje ještě jedna věc, že vždy dva protějších motory (pár (1) a (3) a pár (2) a (4)) se musí otáčet stejným směrem (po směru nebo proti směru hodinových ručiček), jinak by se kvadrokoptéra začala točit kolem osy *roll*, která na obr. 1.3 není vidět, jelikož prostupuje středem kvadrokoptéry (kolmo do obrázku), tzn. v bodě křížení (těžišti kvadrokoptéry) os *yaw* a *pitch*. Právě z tohoto důvodu je při neshodné výkonové křivce všech motorů nemožné použít druhou možnost rozvržení os.

Takže vystává otázka, jaké by mohlo být universální řešení i pro případy, kdy z řídící jednotky půjde identický příkaz do každého ESC, ale každý ESC pošle do svého motoru jiný výkon? Nesouměrnost může být zapříčiněna např. odlišnou délkou vinutí BLDC motoru či nesprávnou prací ESC nebo odlišnými typy ESC. Na první pohled se mi jako nejuniversálnější řešení nabízí zakomponovat do kvadrokoptéry ke každému motoru vlastní otáčkoměr, který posílá hodnoty do řídící jednotky. Dle údaje o otáčkách, které budou vstupovat do virtuálního PID regulátoru, si už každý

PID virtuální regulátor nastaví výstupní hodnotu pro ESC. Toto řešení je ale nevýhodné z pohledu umístění otáčkoměru, kdy magnetický snímač není vhodné řešení díky velkému magnetickému rušení způsobenému motory a optický snímač je zase nevhodné řešení vzhledem k letu např. pod přímím sluncem. Proto je z tohoto pohledu podle mě výhodnější změřit závislost otáček jednotlivých motorů (v laboratorních podmínkách) na výstupních hodnotách řídící desky, které posílá do jednotlivých ESC, sestavit matematickou funkci pro každý motor a do programu tento převod zakomponovat. Ale při použití právě virtuálních PID pro každou osu, lze tento stav ošetřit a prototyp stabilizovat – více v kap. 2.1.

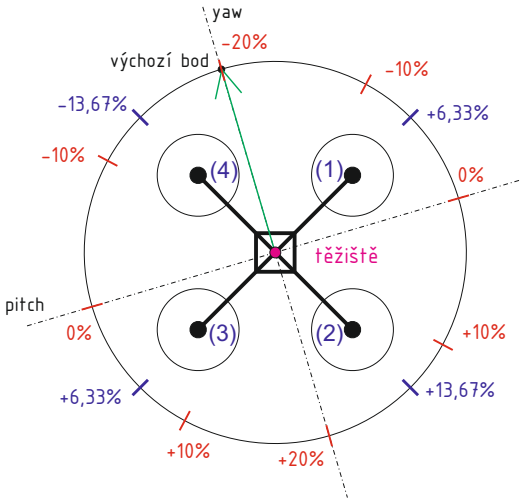
1.1.2 Závislost výkonu motorů na umístění os *yaw* a *pitch*



Obr. 1.4: Grafické znázornění rozdělení výkonů jednotlivým motorům pro metodu 1 a 2

V předchozí podkapitole jsem naznačil jaké rozvržení výkonu jednotlivých motorů bude při použití dvou nejednodušších možností (grafické znázornění viz obr. 1.4). Nyní když bych výchozí bod posunul např. o 16,5° proti směru hodinových ručiček, tak by se pochopitelně musel změnit výkon jednotlivých motorů, aby mi vektor pohybu směřoval od průsečíku os *yaw* a *pitch* (těžiště kvadrokoptéry) do výchozího bodu. Uvažuji ideální motory – na každém motoru identická velikost otáček, tak že se kvadrokoptéra vznáší ve vodorovné poloze v určité výšce, při výchozích otáčkách. Teď chci realizovat let vpřed (tzn. vektor pohybu bude mít směr přímky od průsečíku os *yaw* a *pitch* do výchozího bodu, který je posunut o 16,5° proti směru hodinových ručiček oproti metodě 1) – výchozí výkon jednotlivých motorů se zvýší/sníží o tolik

procent kolik je u každého motoru (modré číslo) – grafické znázornění viz obr. 1.5. Zelená šipka je vektor pohybu vpřed pro danou metodu. Červená čísla jsou výchozí při určování konečného zvýšení/snížení výkonu motorů – lze je nastavit i na více či méně než ±20%, to záleží na nás. Výchozí (červená) čísla jsou umístěna v průsečíku os s kružnicí – těžišti kvadrokoptéry.



Obr. 1.5: Grafické znázornění rozdělení výkonů jednotlivým motorům při posunutí výchozího bodu o 16,5 ° proti směru hodinových ručiček vůči metodě 1

Z obr. 1.5 tedy vyplývá rozdělení pro výkonů. Co znamenají jednotlivé značky jsem řekl výše. U způsobu přerozdělování jde pouze o to, že *yaw* osa má dvě krajní hodnoty námi zadané (ony musí být číselně stejné pouze obrátíme znaménko), přičemž u výchozího bodu je mínusová hodnota, osa *pitch* má nulu. Z toho už vyčteme hodnoty jako z grafu a hodnotu v daném úhlu, kde se nachází motor dopočítáme.

1.1.3 Realizace pomocí zvolených komponent

Rozhodl jsem se, že zvolím způsob rozmístění os podle metody 2 (yaw_2 a $pitch_2$ na obr. 1.3). K tomuto rozhodnutí mě vedla skutečnost, že při metodě 1, PID musí řídit vždy dva motory. Takže je předpoklad, že oba konkrétní motory mají identické výkonové křivky. Což je v praxi nemožné. Ohledně součástí kvadrokoptéry jsem přistoupil ke koupi standardních modelářských komponent určených pro kvadrokoptéry, které jsou k dostání v mnoha modelářských e-shopech a to konkrétně:

- Základní rám – **F450** – jeden z nejrozšířenějších rámů pro stavbu kvadrokoptér, obsahující silové rozvody (rozvody od baterie k ESC) zakomponované přímo v rámu, čímž nám odpadá kabeláž

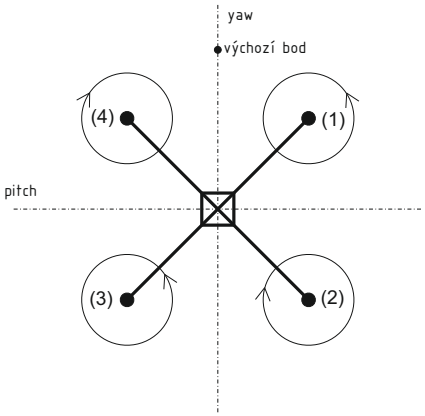
- Motory – **Emax GT2213/935 KV** – normalizované motory pro kvadrokoptéry s 935 KV, maximálním zdvihem 850 g při napájení tří-článkovou lithium-polymerovou baterií (možnost napájení i čtyř-článkovou)
- ESC – **Emax Simon-20 A** – 20 A ESC regulátor s možností nárazového proudu 30 A, programovatelný
- Baterie – **FOXY G2–3EB4030** – tří-článková lithium-polymerová baterie s kapacitou 4500 mAh
- Vysílač a přijímač – **RC souprava Graupner/SJ MZ-10 2,4 GHz HOTT** – 5-ti kanálová RC souprava se šestým kanálem pro plnohodnotnou telemetrii, využívající modulaci FHSS
- Řídící deska – **Arduino Due** – vývojová deska s 32bitovým ARM procesorem a mnoha analogovými a digitálními vstupy/výstupy
- Gyroskop – **Arduino gyroskop s čipem MPU-6050** – plošný spoj osazený čipem obsahujícím gyroskop a akcelerometr s vývody pro připojení na sběrnici I^2C, respektive sběrnici TWI
- Ultrazvukový senzor vzdálenosti – **Arduino měřič vzdálenosti ultrazvukový HC-SR04** – měření vzdálenosti od 2 cm do 450 cm s přesností až 3 mm

Hmotnost jednotlivých komponent viz tab. 1.1. Základní rám by měl být dle údajů od výrobce dimenzovaný na 1600 g, což mi zatím dostačuje a teprve v budoucnu uvidím nakolik bude reálná instalace např. kamery nebo dalších senzorů. Se zdvihem motorů by problém být ale neměl, jelikož výrobce udává maximální zdvih 850 g, což nám dává dohromady maximální zdvih všech čtyř motorů 3400 g.

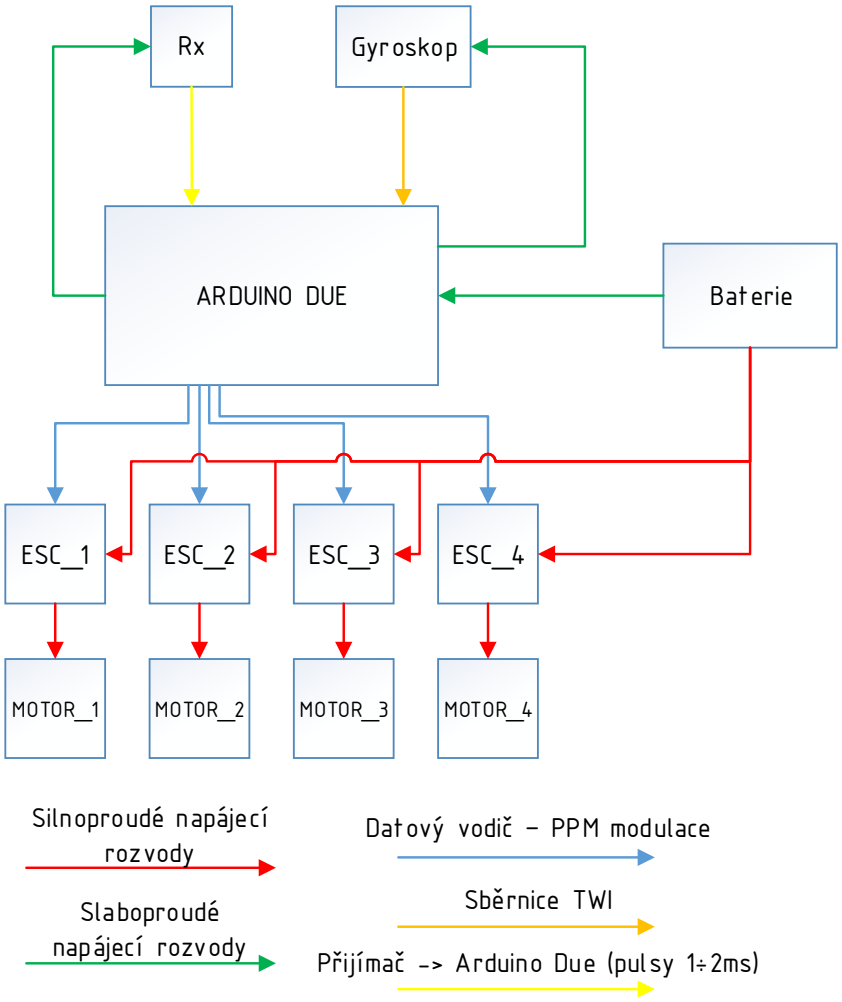
Tab. 1.1: Hmotnost jednotlivých komponent

Komponenta	**Hmotnost**
Základní rám	282 g
Motory	4×55 g
ESC	4×28 g
Baterie	320 g
Přijímač	10 g
Řídící deska	36 g
Gyroskop	3 g
Ultrazvukový senzor	$8 \times 8{,}5$ g
Kabeláž a ostatní	cca 50 g
Celkem	cca 1101 g

Ještě uvedu očíslování motorů, jak jsem si je očísloval a směr jejich rotace, protože to ušetří budoucí práci a popis prototypu – viz obr. 1.6.



Obr. 1.6: Očíslování a směr rotace motorů prototypu



Obr. 1.7: Blokové schéma prototypu

Z výše uvedených komponent bude složen můj prototyp. Většina komponentů je částečně normalizována, tak aby do sebe zapadly – např. stejná velikost závitů, rozteč mezi jednotlivými dírami apod. Celkové blokové schéma zapojení i s legendou viz obr. 1.7.

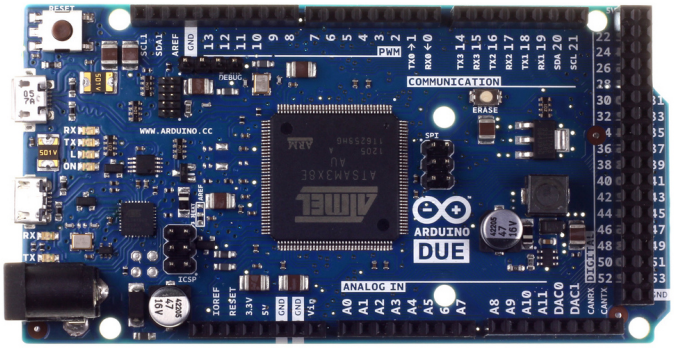
Součinnost jednotlivých komponent

Nyní stručně nastíním součinnost jednotlivých komponent mého prototypu. Informace o změnách letu, tím myslím pokyny typy leť vpřed, leť nahoru, leť dolů apod., zadáváme do pákového vysílače, který je spárován s přijímačem a hodnoty následně převáděny na PPM rámec, obsahující informaci o všech kanálech. Komunikace probíhá v nelicencovaném pásmu 2,4 GHz. Přijímač je napájen pomocí desky Arduino Due, která na jednom pinu poskytuje stabilizované napájecí napětí 5 V – lze tedy přijímač napájet přímo z řídící desky, jelikož přijímač nemá závratný proudový odběr a tudíž příliš nezatěžuje desku. Přijímač vysílá do řídící desky pulsy v délce 1–2 ms (rozložený PPM rámec) – délka jednotlivých pulsů odpovídá velikosti vychýlení jednotlivých pák na vysílači. Arduino impulsy z přijímače zpracuje a jednoduše řečeno převede na hodnotu srozumitelnou pro ESC regulátory. K této hodnotě se ještě připojí hodnota z virtuálních PID regulátorů, které kvadrokoptéru stabilizují pomocí hodnot náklonu v jednotlivých osách. Tyto hodnoty náklonu získává Arduino Due od Arduino gyroskopu skrz sběrnici TWI. Následně Arduino Due vyšle PPM signál k jednotlivým ESC regulátorům, nesoucí výslednou hodnotu pro konkrétní motor. ESC regulátor informaci převede na výkonové impulsy, které následně posílá do svého motoru. Na podrobnější popis se podíváme v jednotlivých kapitolách.

1.2 Arduino Due

Arduino Due je jedna z mnoha vývojových desek od společnosti Arduino, která ale svým výkonem a parametry ostatní desky (až na jednu vyjímku – Arduino Ýun s jádrem AR9331 Linux) v mnohém výkonnostně převyšuje. Základní práce s Arduinem Due se nijak neliší od ostatních Arduino desek a i pro něj platí obecná politika Arduina, že se jedná tzv. „open-source“ projekt. Jako drtivá většina Arduin tvoří srdce desky procesor od firmy Atmel a i pro desku Arduino Due jako takovou je typické jednotné Arduino zabarvení. Ilustrační fotografie desky Arduino Due viz obr.1.8. [2]

Jako první Arduino je vybavené jádrem Atmel SAM3X8E ARM Cortex-M3 CPU. Jako vůbec první vývojová deska od Arduina obsahuje 32bitový ARM procesor. Na rozdíl od většiny vývojových desek od Arduina, které operují na 5 V, Arduino



Obr. 1.8: Arduino Due

Due používá operační napětí 3,3 V, tzn. vstupní napětí na pinech nesmí přesáhnout 3,3 V (ekvivalentně Vám Arduino Due neposkytne na výstupu napětí vyšší než 3,3 V). Ovšem díky pinu *IOREF*, kam lze přivést referenční napětí 5 V, lze toto omezení obejít a připojit i nástavby (tzv. *shieldy*), které pracují na napětí 5 V. Co se týče I/O, je deska Arduino Due vybavena 54 digitálními piny, které lze použít jako vstupy nebo jako výstupy (deklarace se provádí až ve vývojovém prostředí), z nichž 12 poskytuje výstup pro PWM, případně PPM modulaci. Dále obsahuje 12 analogových vstupů, 4× UART rozhraní, 2× DA převodník (analogové výstupy), 2× TWI rozhraní, 1× SPI přijímač, 1× JTAG rozhraní, 1× napájecí konektor a 2× micro USB port – jeden nativní „native“, sloužící pro připojení zařízení (myši, klávesnice, telefony…) a druhý programovací „programming“ k nahrávání řídícího kódu. Procesor je taktován na 84 MHz, což zajišťuje krystalový oscilátor. Za zmínku na úvod ještě stojí, že deska disponuje vlastností USB OTG. Toť stručné shrnutí parametrů/vlastností vývojové desky Arduino Due. Soupis již zmíněných a ostatních základních parametrů/funkcí viz tab. 1.2. [2]

V porovnání s ostatními deskami od Arduina spatřuji největší výhodou, že obsahuje 32bitový ARM CPU tzn. v jednom taktu zvládne zpracovat 4 B širokou zprávu, při taktovací frekvenci 84 MHz. Mezi další výhody patří, v porovnání s ostatními deskami od Arduina, 96 kB velká SRAM paměť a 512 kB flash paměť. Stručné porovnání s vybranými deskami od Arduina viz tab. 1.3. [3]

Tab. 1.2: Základní paramtery Arduina Due

Mikročip	AT91SAM3X8E
Pracovní napětí	3,3 V
Doporučené napájecí napětí	7-12 V
Limity napájecího napětí	6-16 V
Počet digitálních I/O pinů	54 (12 z nich umožňuje PWM)
Počet vstupních analogových pinů	12
Počet výstupních analogových pinů	2 (2× DA převodník)
Max. SS proud na jeden 3,3 V pin	800 mA
Max. SS proud na jeden 5 V pin	800 mA
Flash paměť	512 kB
SRAM	96 kB
Pracovní frekvence	84 MHz
Délka	101,52 mm
Šířka	53,3 mm
Váha	36 g

Tab. 1.3: Porovnání vybraných vývojových desek Arduino

Typ desky	Procesor	Pracovní/vstupní napětí	Takt jádra [MHz]	Analogové I/O
Due	ATSAM3X8E	3.3 V / 7-12 V	84	12/2
Leonardo	ATmega32U4	5 V / 7-12 V	16	12/0
LilyPad	ATmega168V	2,7-5,5 V / 2,7-5,5 V	8	6/0
Mega 2560	ATmega2560	5 V / 7-12 V	16	16/0
Nano	ATmega168	5 V / 7-9 V	16	8/0
Uno	ATmega328P	5 V / 7-12 V	16	6/0

Typ desky	Digitální I/O / PWM	EEPROM [kB]	SRAM [kB]	Flash paměť [kB]
Due	54/12	–	96	512
Leonardo	20/7	1	2,5	32
LilyPad	20/7	1	2,5	32
Mega 2560	54/15	4	8	256
Nano	14/6	0,512	1	16
Uno	14/6	1	2	32

1.2.1 Architektura ARM

Tzv. ARM – *Advanced RISC Machine* – je architektura procesorů, jak již z anglického názvu vyplývá, spadající do kategorie RISC procesorů vyvíjených Britskou společností ARM Holdings.

Procesor Cortex-M3

Vedoucí třída 32bitových ARM procesorů pro deterministické a tzv. real-time aplikace (aplikace pracující v reálném čase) a první z řady z procesorů ARM založených na technologii ARMv7. Přednostně vyvinuty pro široké spektrum použití, čehož je docíleno kompromisem mezi poměrně vysokým výpočetním výkonem a nízkonákladovými požadavky. Patří sem využití např. jako mikrokontrolery pro automobilový průmysl, průmyslové kontrolní a řídící systémy, síťová technika, senzory atd. Procesory z této rodiny jsou též lehce a rychle konfigurovatelné a lze je využít i pro implementace, kde se vyžaduje tzv. ochrana paměti, to je předurčuje i v použití pro citlivá zařízení na minimálním prostoru. [4]

Jak již bylo výše zmíněno, třída Cortex-M3 je založena na technologii ARMv7, která se ale dále dělí na 3 profily – každý profil má trochu jiné vlastnosti: [5]

- Profil A (ARMv7-A) – pro komplexní aplikace potřebující výkonný procesor a podporu virtuální paměti jednotkou správy paměti jako je např. Windows Embedded, případně pro Java aplikace vyžadující specifickou hardwarovou podporu
- Profil B (ARMv7-B) – především pro real-time systémy vyžadující vysoký výpočetní výkon a krátkou dobu reakce
- Profil C (ARMv7-C) – pro mikrokontrolery, u kterých je hlavním kritériem je nízká cena a spotřeba energie, případně pak rychlá odezva na přerušení

1.2.2 Sběrnice TWI

Zkratka vychází z anglického slovního spojení – *Two Wire Interface*. Jedná se o téměř identickou sběrnici jakou je I^2C sběrnice, se kterou je zároveň kompatibilní. K vytvoření TWI sběrnice přispěla např. firma Atmel (a několik dalších společností), aby se vyhnuli problému s ochranou známkou na sběrnici I^2C.

Jde o dvouvodičovou sériovou sběrnici pracující na principu tzv. „MASTER-SLAVE“, kdy jedno zařízení je *MASTER* a ostatní jsou *SLAVE*. I když se jedná o sériovou komunikaci, používají se dva vodiče (kanály):

- 1. vodič – SDA – z anglického *Synchronous DAta*, zde probíhá veškerá komunikace po lince

- 2. vodič – SCL – z anglického *Synchronous CLock*, *MASTER* zařízení generuje hodinový signál a posílá po SCL vodiči

Každé zařízení připojené do sběrnice má svoji adresu. Komunikace probíhá tak, že pokud si *MASTER* zařízení přeje začít komunikaci, nejdříve vyšle po SDA adresu zařízení, se kterým chce komunikovat. Naslouchají všechna připojená zařízení, ale pouze zařízení s požadovanou adresou potvrdí přijetí ACK. Součástí adresy zařízení je i bit, kterým *MASTER* zařízení říká, kterým směrem se bude komunikovat, tzn. kterým směrem se budou přenášet data, buď ve směru *MASTER* → *SLAVE* (binární nula) nebo *SLAVE* → *MASTER* (binární jednička). Toto opatření je nutné z toho důvodu, že sběrnice TWI umožňuje pouze poloduplexní komunikaci. [6]

Komunikační protokol

Nyní si uvedeme jako příklad naznačení komunikace po lince TWI způsob čtení dat ze SLAVE zařízení viz obr.1.9 a zápis dat do SLAVE zařízení viz obr. 1.10. Legenda k obr. 1.9 a obr. 1.10, viz obr. 1.11. Z obr.1.9 a obr.1.10 je patrné, že každý přenos začíná START sekvencí a končí STOP sekvencí. [6]

S	7bitová adresa SLAVE zařízení	R	A	DATA	A	DATA	–A	P

Obr. 1.9: Příklad komunikace při čtení dat ze SLAVE

S	7bitová adresa SLAVE zařízení	W	A	DATA	A	DATA	–A	P

Obr. 1.10: Příklad komunikace pro zápis dat do SLAVE

S	START sekvence
R	čtení dat ze SLAVE (H úroveň)
W	zápis dat do SLAVE (L úroveň)
A	potvrzení příjmu ACK (L úroveň)
–A	nepřišlo potvrzení příjmu (H úroveň)
P	STOP sekvence
adresa i data přenášeny od MSB po LSB	

Obr. 1.11: Legenda k obr. 1.9 a obr. 1.10

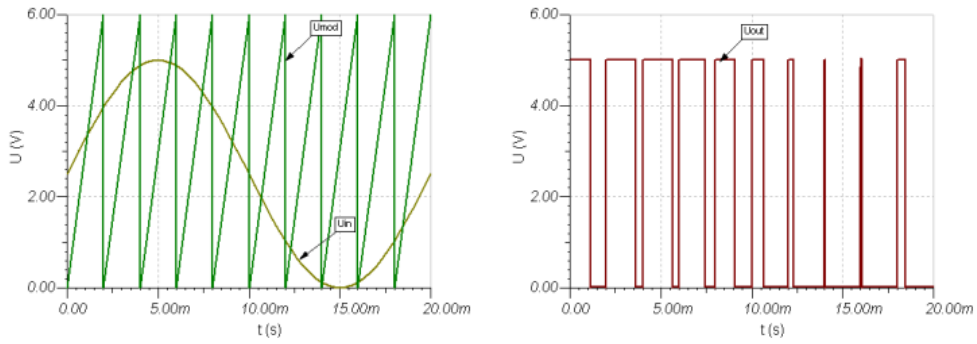
1.2.3 PWM a PPM modulace

PWM modulace

Pulsně šířková modulace PWM, z anglického spojení – Pulse Width Modulation. Jde o jednoduchý a hojně využívaný způsob modulace pro přenos dat a v podstatě i výkonu. Celý princip je založen na měnění střídy obdélníkového signálu s konstantní periodou. Kde střída (taky se lze setkat s anglickým vyjádřením, u nás též velmi zažitým, „duty cycle“) udává poměr mezi trváním nízké a vysoké úrovně(při standardní funkci nenajdeme jinou úroveň). Střídu lze udávat ve dvou vyjádřeních: [7]

- Absolutní míra – např. 1:4, kde číslo 1 značí délku trvání vysoké úrovně a číslo 4 značí periodu (v podstatě jinak řečeno poměrové vyjádření)
- Procentuální míra – např. střída 35 % značí, že 35 % času z celkového trvání periody, je signál ve vysoké úrovni

Největší uplatnění nachází např. při řízení stejnosměrných motorů, kdy je nepraktické pro regulaci otáček neustále snižovat/zvyšovat napětí za účelem docílení požadovaných otáček. V podstatě analogicky střídání vysoké a nízké úrovně si lze představit jako nuly a jedničky, tzn. že tvorba PWM signálu je velmi jednoduchá, protože postačí pouze přepínat mezi vysokou a nízkou úrovní v odpovídajících časových intervalech. Modulátor vytvářející takové průběhy lze jednoduše vytvořit pomocí komparátoru, kde na jeden vstup přivádíme pilový (na obr. 1.12 jako nosný signál) a na druhý vstup signál sinusový (obr. 1.12) jako modulovaný/datový signál. Komparátor následně překlápí v závislosti na poměru obou signálů výstupní úroveň. [7]



Obr. 1.12: Příklad jednoduché tvorby PWM signálu, převzato z [8]

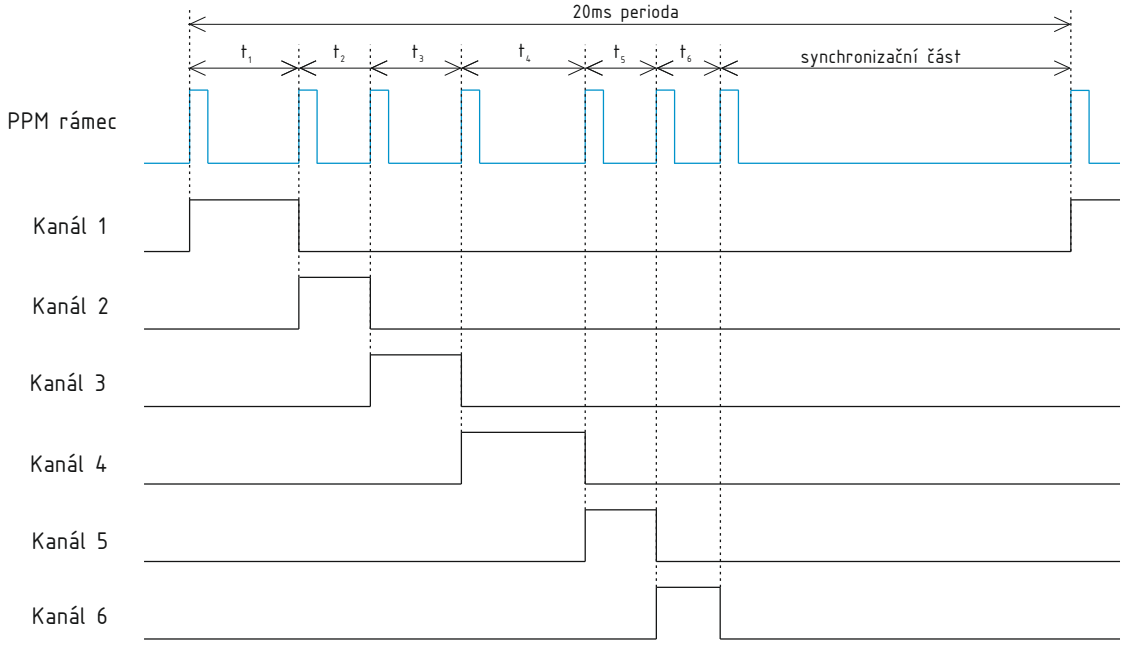
PPM modulace

Tzv. pulsně polohová modulace, z anglického „Pulse Position Modulation“. PPM pracuje na principu zakódování hodnoty jako polohy impulsu ve vymezeném časo-

vém intervalu. V modelářství se tato modulace hojně používá u RC souprav (jak u vysílače, tak i přijímače), ale hlavně v optických komunikačních systémech. Je to vedle PWM a PAM (Pulsně Amplitudová Modulace) další z rodiny nekvantovaných impulsových modulací. Vyslání osamoceného impulsu v odpovídající časové poloze rámce vyjadřuje n-bitovou informaci – obsahuje celkem 2^n možných časových poloh. [9]

PPM rámec používaný v RC soupravách Na obr. 1.13 vidíme názorný příklad PPM rámce (první řádek – „Received PPM Frame“). Jednoduše tedy řečeno, PPM spočívá v posílání jednotlivých 300 μs dlouhých obdélníkových impulsů. Z toho se zakódovaná hodnota vyčte ze vzdálenosti nástupních hran jednotlivých pulsů. Na obr. 1.13 je v prvním řádku zobrazen PPM rámec, kde je zakódováno 6 kanálů RC soupravy (může jich být až 8) a následně níže naznačen rozklad PPM rámce na jednotlivé kanály.

V dnešní době je PPM modulace a potažmo PPM rámec, již jakýsi standart používaný RC soupravami. Je tu sice možnost, že by se jednotlivé 300 μs pulsy mohly krýt při posílání menších časových hodnot než je 300 μs, ale to je ošetřeno tím, že RC soupravy používají rozsah hodnot pro páky 1–2 ms.



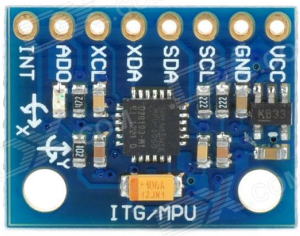
Obr. 1.13: Ukázka rozkladu PPM rámce na jednotlivé kanály

1.3 Arduino gyroskop

Při volbě gyroskopu byl originální Arduino gyroskop s čipem MPU-6050 jasnou volbou právě zejména z hlediska, že obsahuje čip MPU-6050, který vrací již přímo náklony v jednotlivých osách, respektive akceleraci ve gramech a náklon ve stupních a nemusíme řešit další přepočty v samotné řídící desce. Dalším kladem je, že Arduino gyroskop obsahuje možnost přímého připojení na sběrnici I^2C (respektive TWI). Je vybaven tří-osým gyroskopem a akcelerometrem a DMP („Digital Motion Processor“), umožňující provádět konečné výpočty a posílat hodnoty náklonu ve stupních nebo zrychlení ve gramech. Dále obsahuje teploměr, který je téměř nepoužitelný jelikož nevrací přesnou teplotu okolí jako spíše nějakou teplotu mezi teplotou čipu a okolí. Ještě lze zmínit, že obsahuje možnost připojení externího magnetometru a kromě pinů pro připojení napájení nebo sběrnice, obsahuje deska ještě pin pro připojení přerušení na řídící desku. Ilustrační obrázek Arduino gyroskopu viz obr 1.14. Souhrn paramterů viz tab. 1.4. [11]

1.3.1 MPU-6050

Čip MPU-6050 se vyznačuje nízkou spotřebou a nízkou cenou, což ho předurčuje k použití např. v chytrých telefonech a tabletech. Jak jsme si už řekli, obsahuje tříosý akcelerometr a gyroskop a DMP zpracovávající komplexní šesti-osé MotionFusion algoritmy. Lze i nastavit citlivost akcelerometru a gyroskopu.



Obr. 1.14: Arduino gyroskop

Od MPU-6050 můžeme získávat čisté hodnoty ve stupních a gramech, ale jde pracovat i s tzv. „surovými hodnotami“, tedy ještě hodnotami nepřepočítanými pomocí DMP. Navíc existuje knihovna právě pro komunikaci Arduina Due s MPU-6050, tudíž je jeho používání v platformě Arduino více než pohodlné. Čip samotný umožňuje komunikaci i po SPI sběrnici, ale deska Arduino, kde je zasazen umožňuje pouze komunikaci po sběrnici I^2C (respektive TWI). Operační proudový odběr gyroskopu se pohybuje kolem 3,6 mA (5 μA ve stand-by módu), akcelerometru 500 μA (proud stand-by módu se liší v závislosti na frekvenci vzorkování). Zajímavostí a zároveň užitečnou vlastností je, že DMP dokáže do svých výpočtů zahrnout i externě

připojený magnetometr. Dále z hlavních vlastností lze ještě zmínit, že má programovatelné přerušení. [12]

Tab. 1.4: Parametry Arduino gyroskopu

Použitý čip	MPU-6050
Napájení	3 – 5 V
Komunikace s okolím	I^2C
AD převodník	16bitový (16bitové výstupní slovo)
Rozsah gyroskopu [°/s]	+250, +500, +1000, +2000
Rozsah akcelerometru [g]	±2, ±4, ±8, ±16
Rozměry [mm]	$21 \times 15 \times 1{,}2$
Hmotnost [g]	3

1.4 Pohon kvadrokoptéry

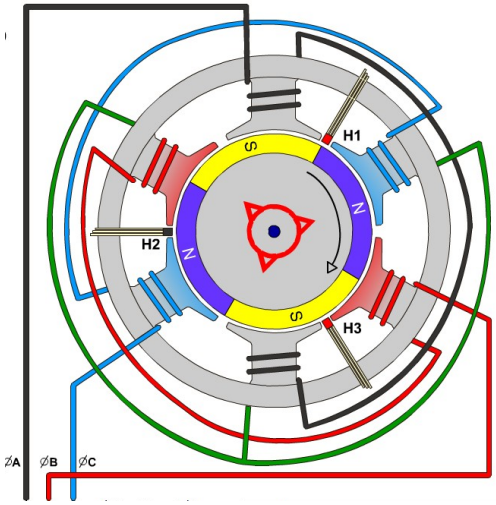
Vzhledem k povaze zadání – orientace spíše na řídící algoritmus a práci s Arduinem Due, využiji standardní modelářské komponenty určené pro kvadrokoptéry. Tzn. byly použity bezkomutátorové motory, příslušné fyzické regulátory (ESC) určené podle proudového odběru použitých motorů a lithium-polymerový akumulátor.

1.4.1 Bezkomutátorový motor – BLDC

Nebo-li bezkartáčový, anglicky tzv. „brushless“ motor, zkráceně BLDC. Principem funkce se dost podobá krokovým motorům. Patří do skupiny stejnosměrných synchronních motorů, běžně v praxi označovaný jako *střídavé*. Jak již název napovídá, motory tohoto typu neobsahují kartáče pro komutaci napětí. Komutace je tedy zajištěna elektronicky, v případě mého projektu komutaci a obecné řízení otáček motoru poskytuje ESC regulátor obsahující veškerou nezbytnou řídící elektroniku. V mém případě je rotorem těchto motorů plášť, obsahující permanentní magnety otáčející se kolem pevně umístěného statoru, který obsahuje odpovídající počet cívek. Toto uspořádání se nazývá *vnější*, analogicky lze popsat i *vnitřní* typ, kdy je plášť stator obsahující cívky a rotor uvnitř pláště je složený z permanentních magnetů. Podle počtu n permanentních magnetů rozlišujeme n-pólové motory. Lze se setkat i s jedno nebo dvou fázovými BLDC, ovšem nejvíce se využívají tří-fázové. [13]

Řízení BLDC

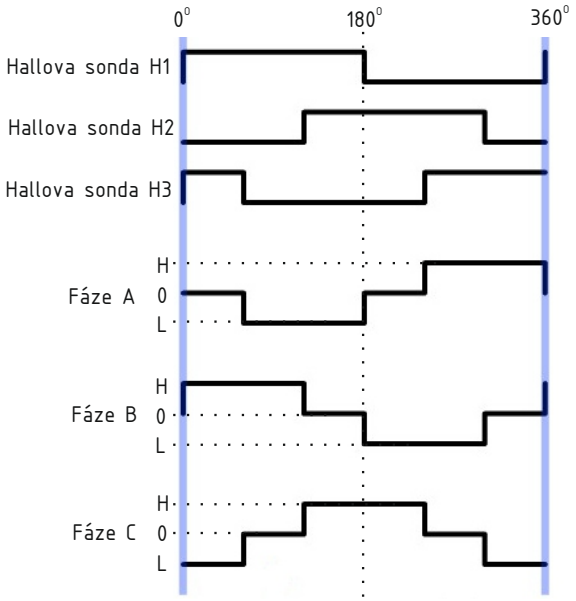
Pro řízení bezkomutátorového motoru je nezbytné znát polohu rotoru v daném okamžiku. To nejčastěji zajišťuje trojice hallových sond využívající hallova jevu, rozprostřené po 60° nebo po 120°. Názorné propojení 6-ti pólů a rozmístění tří hallových sond ve verzi posunutí o 120° viz obr. 1.15.



Obr. 1.15: Rozmístění hallových sond a zapojení vnitřního čtyř-pólového BLDC, převzato z [14]

Metoda řízení pomocí senzorů – např. pomocí hallových sond, viz obr. 1.16, který odpovídá příkladu řízení pro obr. 1.15 – je jeden způsob řízení a využívá se v náročnějších aplikacích, které požadují přesné informace o poloze rotoru. Nevýhodou je nutnost dodatečného místa, další pájení a tím roste samozřejmě i cena. Princip je takový, že pokud se blízkosti hallovy sondy objeví magnetické pole dochází ke generování kladného nebo záporného napětí, což závisí na polaritě magnetického pole (jižní/severní pól). [13]

Řízení při absenci senzorů polohy natočení rotoru se hodí spíše pro méně náročnější aplikace nebo v případech, kdy je možnost např. znečištění hallových sond, tzn. senzory by nedodávaly adekvátní hodnoty. Výhodou naopak je, že se k motoru nemusí připojovat další čidla za účelem určování polohy rotoru. Princip tedy spočívá v měření napětí naindukovaného ve vinutí, které není v daném komutačním kroku připojeno k napájení. Toto napětí se nazývá *zpětná elektromotorická síla* (zkráceně EMF) a informaci o poloze rotoru lze vyčíst z jeho velikosti. Ale aby jsme vůbec mohli toto zpětné na-indukované napětí vůbec detekovat, musíme nejprve rotor roztočit do určitých minimálních otáček, kdy lze už toto napětí detekovat. [13]



Obr. 1.16: Grafické znázornění komutační logiky pro snímače rozmístěné po 120°

BLDC Emax 2213/935KV

Ilustrační foto viz obr. 1.17, parametry viz. tab. 1.5. Typ Emax 2213/935KV jsem si zvolil v závislosti na ceně a recenzí, z nichž vyplynulo, že se jedná o průměrný motor s dobrou cenou, plně dostačující mým potřebám.

Ještě je třeba říct, že jsem vybral deseti palcové vrtule vrtule. Obecně platí, že čím menší vrtule, tím horší stabilizace, ale lepší manévrovatelnost. Deseti palcové vrtule patří k těm větším a zvolil jsem je kvůli tomu, aby se kvadrokoptéra dobře stabilizovala, ale zároveň nebyl velký proudový odběr, což by u ještě větších vrtulí byl problém. Ještě nutno říci, že písmena KV v názvu kvadrokoptéry jsou jednotka vyjadřující počet otáček na jeden volt. Hodnota 935 KV je průměrná pro rekreační létaní nebo pořizování záběrů, ale u akrobatických kvadrokoptér se setkáváme obyčejně s hodnotami vyššími než 1500 KV.

Obr. 1.17: BLDC Emax 2213/935KV

Tab. 1.5: BLDC 2213/935KV parametry

KV	935
Hmotnost	55 g
Průměr motoru	27,9 mm
Výška motoru (i s hřídelí)	39,7 mm
Napájení	3 nebo 4 Li-Pol články
Maximální zdvih	750 g
Maximální zatížení	16 A
Maximální zdvih	750 g
Průměr hřídele	6 mm
Doporučený ESC	18 A

1.4.2 ESC regulátor

Z anglického *Electronic Speed Controller*. Stručně řečeno, součástka, která přijímá signál v nějakém tvaru (v mém případě PPM modulovaný signál) a na základě přijmu-té hodnoty, ovládá otáčky BLDC motoru. ESC již obsahuje veškerou elektroniku nutnou k řízení, takže stačí pouze připojit napájecí kabely k baterii – **nutné rozlišit polaritu, jinak hrozí zničení** –, datové kabely k řídící jednotce (kladná polarita, záporná polarita a datový vodič) a nakonec tři barevně nerozlišené silové vodiče do motoru. U vodičů směřujících do motoru není nutné rozlišovat zapojení – motor se bude pouze točit po směru nebo proti směru hodinových ručiček a na případnou následnou změnu směru otáčení stačí přepojit pouze dva z kabelů.

Podstatnou součástí dnes už většiny ESC regulátorů je tzv. BEC obvod – z anglického *Battery Eliminator Circuit*. Je nutný z hlediska modelářských potřeb, kdy by bylo nepraktické používat dvě baterie – jednu např. tří článkovou li-pol pro napájení silových obvodů (12,6 V) a druhou s menším napětím k napájení řídících obvodů. S BEC obvodem stačí ESC připojit jednu několika článkovou (většinou tří až čtyř) lithium-polymerovou baterii (prakticky se dnes jiné nepoužívají), která primárně napájí silové obvody. Jelikož ESC potřebuje pro silovou část obyčejně minimálně tří až čtyř článkovou baterii, což je 12,6 V, nemůže baterie napájet i řídící elektroniku, jejíž napájecí napětí se pohybuje v rozmezí tři až šest voltů – byla by tudíž potřeba další baterie pro řídící elektroniku. Toto nám řeší BEC obvod, který velké napájecí napětí z hlavní baterie stabilizuje většinou na 5 V a poskytuje ho pro napájení řídící elektroniky – trojice vodičů s nejmenším průřezem vedoucích z ESC, ze kterých je právě jeden kladné polarity napětí, druhý záporné polarity napětí výstupního stabilizovaného napětí BEC obvodu a třetí vodič je datový. To je dnes zažitý standard modelářských potřeb, kdy právě pomocí BEC je napájena řídící předprogramovaná

jednotka či přijímač. Jenže já jako řídící jednotku používám Arduino Due, které ale musí být napájena samostatně a nemá problém s napájecím napětím třech článků lithium-polymerové baterie.Takže řešení v mém případě je, že BEC výstupní napětí vůbec nevyužiji, jelikož Arduino Due je napájeno přímo z baterie a přijímač z Arduina Due, pomocí pinů poskytujících stabilizované napájecí napětí. Takže BEC napájecí vodič kladného napětí vůbec nepřipojuji, jelikož by mohlo dojít k poškození Arduina Due, protože napětí na pinech je u Arduina Due pouze 3,3 V – nechám ho jednoduše odpojen.

Ještě lze zmínit, že dnes již téměř každý ESC obsahuje detekci baterie, tzn. dokáže identifikovat počet článků lithium-polymerové baterie a podle toho si nastavit parametry, dále pak identifikaci poklesu napětí článků baterie apod.

ESC Emax Simon 20A

Zvolil jsem ESC Emax Simon 20A, jeden ze zástupců levnějších 20 A regulátorů, který ale plně dostačuje mým nárokům – max. trvalý proud pro mé motory je 16 A, takže mám při volbě 20 A ESC ještě dostatečou rezervu. Souhrn základních parametrů viz tab. 1.6 a ilustrační foto regulátoru Emax Simon 20A viz obr. 1.18. [15]

Tento a většina jiných ESC je dnes již programovatelných. Některé pomocí přímo programovací karty nebo pomocí nastavování pák na vysílači (můj typ ESC umožňuje obě možnosti), v mém případě pomocí výstupních hodnot Arduina Due – např. standardní spouštěcí proces je, že musíme stáhnout páku plynu (výstup Arduina) do nejnižší polohy, připojit řídící jednotku a pak teprve připojit hlavní baterii. ESC nás postupně o průběhu informuje pomocí jednoduchých zvukových signálů. Obdobné postupy platí i např. pro nastavování právě maximální a minimální hodnoty výstupu ESC. Dále ještě lze nastavit (z mnoha dalších možností) např. odpojení nebo pokles na určitou hodnotu výstupu pro motor, při určitém poklesu napájecího napětí, nastavení určité hodnoty výkonu při ztrátě signálu, kontrolní frekvenci, definovat „nízké napětí“, plynové křivky pomocí programovací karty atd. [15]

Obr. 1.18: ESC Emax Simon 20A, převzato z [16]

Tab. 1.6: ESC Emax Simon 20 A

Maximální trvalý proud	20 A
Nárazový proud	25 A
Počet Li-Pol článků	2 – 3
Rozměry [mm]	$52 \times 26 \times 7$
Hmostnost	28 g
BEC mód	Lineární
BEC výstup	2 A/5 V
Programovatelný	Ano

1.4.3 Lithium-polymerová baterie

Zkráceně tzv. „Li-Pol“. Jedná se o relativně nový druh baterie, ale dnes používaný v mnoha zařízeních (fotoaparáty, mobilní telefony, tablety, notebooky, RC modely, elektro-automobily…). Byly vyvinuty ze starších lithium-iontových akumulátorů („Li-Ion“) a v podstatě především zlepšují jejich vlastnosti. Lze říci, že lithium-polymerové akumulátory dnes představují nejlepší komerčně vyráběnou variantu pro skladování elektrické energie. První li-poly se na trh dostaly již v roce 1996 a to již pět let po svých předchůdcích li-ionech. S těmi se nejvíce liší v kapacitě, která je zhruba o 10 %větší než u li-ionů (na jednotku váhy), až o 15 % lehčí, ale až o 20 % objemnější a s postupem času ztrácí kapacitu poněkud rychleji než li-iony, což jsou, ale oproti NiCd nebo NiMh článkům stále zanedbatelné ztráty kapacity. [17]

Největší předností lithium-polymerových článků je poměrově velmi vysoká hustota uchovávané energie, tzn. hmotnostně stejné starší typy baterií uchovaly při stejné váze článku mnohem méně elektrické energie následně druhým největším kladem je, že li-poly jsou bez tzv. „paměťového efektu“ – nemusíme je plně vybít, aby jsme je mohli plně nabít. V neposlední řadě spousty výhod ještě figuruje velmi malý samovybíjecí proud, pohybující se okolo 1% celkové kapacity za měsíc (u NiCd akumulátorů to může být až 15% za měsíc). Naopak z nevýhod se nejvíce projevuje, že při nabíjení a vybíjení nesmí být překročeny určité hodnoty, jinak hrozí nevratné poškození – proto je k nabíjení nutné mít speciální drahé nabíječe k tomu určené. Dále jsou velmi křehké a citlivé (zranitelné), tzn. při poškození obalu je velké riziko požáru nebo poškození zdraví. [17]

Všechna napětí se pohybují v závislosti na použité technologii – jmenovité napětí jednoho článku se pohybuje v rozmezí 3,3–3,7 V, vybitý článek někde mezi 2,7–3 V (neměl by klesnout níž, jinak hrozí zničení) a plně nabitý článek 4,2–4,35 V (též by tuto hodnotu neměl přesáhnout – o toto se právě stará speciální nabíječka). [17]

FOXY G2–3EB4030

Vybral jsem tří-článkovou lithium-polymerovou baterii FOXY G2–3EB4030 s kapacitou 4500 mAh. Doporučený nabíjecí proud se pohybuje v rozmezí 1–2 *C*, dlouhodobý vybíjecí proud se udává maximálně 40 C a špičkový vybíjecí proud může být až 80 C. Kromě vodičů s rozlišenou polaritou obsahuje také trojici dalších vodičů určených k připojení na speciální nabíječ, jelikož nabíječ potřebuje dostávat zpětnou vazbu z baterie o stavu všech nabíjených článků. Svými parametry se tento typ baterie pěkně hodí k mé aplikaci a odhadovaná výdrž na jedno nabití by měla být 15-20 min. Do jaké míry jsou mé odhady správné, experimentálně ověřím při letu-schopném modelu. Ilustrační foto baterie FOXY G2–3EB4030 viz obr. 1.19. [18]

Obr. 1.19: Lithium-polymerová baterie FOXY G2–3EB4030

1.5 Vysílač, přijímač

Při volbě vysílače a přijímače jsem opět přikročil ke koupi standardního vybavení a po přečtení několika recenzí a diskuzí jsem zvolil RC soupravu Graupner/SJ MZ-10 2,4 GHz HOTT. Jedná se v podstatě o 5-ti kanálovou RC soupravu obsahující i šestý kanál pro plnohodnotný přenos telemetrie. Obsahuje kuličková ložiska v obou křížových ovladačích, možnost pomocí šroubováku upravovat citlivost jednotlivých pák, pak zda se páky budou pomocí pružiny vracet nebo zůstanou v nastavené poloze apod. Jako napájení slouží 4 alkalické baterie a je zde možnost připojení napájecího konektoru a napájet se pomocí zdroje. Umožňuje fuknce jako žák-učitel, ovládání současně teoreticky až 200 RC souprav najednou a spoustu dalších funkcí, které ke svému účelu nepotřebuji. Vysílač viz obr. 1.20. [19]

Vysílač s označením MZ-10 je již z výroby spárován s přijímačem Hott GR-12L. Teoretický maximální dosah se udává 4 000 m. Jako modulace je použita FHSS – *Frequency-Hopping Spread Spectrum* – díky které je zajištěna vysoká spolehlivost. Přijímač je dělaný tak, aby byl napájen pomocí BEC obvodu, ale já využiji napájení 5 V na výstupu Arduina Due. [19]

Obr. 1.20: Vysílač RC soupravy Graupner/SJ MZ-10 2,4 GHz HOTT, převzato z [20]

1.5.1 FHSS modulace

Princip spočívá v rychlém přepínání nosných rádiových signálů mezi mnoha kmitočty v určitém pásmu – přepnutí na další kmitočet je založeno na pseudo-náhodné sekvenci a je známo jak vysílači, tak přijímači. Toho využívá několikanásobná přístupová metoda k zařízení označována FH-CDMA – *Frequency-Hopping Code Division Multiple Access*. Má souprava Graupner/SJ MZ-10 2,4 GHz HOTT využívá pásmo 2 400–2 483,5 MHz. [21]

Výhodami tedy jsou, že každé kmitočtové pásmo je rozděleno na několik podpásem, mezi kterými se právě nosný kmitočet mění, tzn. že interference na určitém kmitočtu ovlivní pouze ten určitý kmitočet, ale díky rychlému přeskakování mezi kmitočty, se následně už přeskočí na další kmitočet a interference se již neprojeví. Signály se široce rozprostřeným spektrem jsou díky této modulaci vysoce odolné vůči úzkopásmovému rušení. FHSS signály je taky těžké odposlouchávat, jelikož neustále mění kmitočet a lze v jednom pásmu provozovat mnoho zařízení, protože díky pseudo-náhodné sekvenci při velmi rychlém přeskakování kmitočtů, je zde malá šance, že dvě zařízení se připojí na jeden stejný kmitočet. Např. ve vojenském použití tyto výhody ještě znásobí robustní šifrování. [21]

1.6 Ostatní komponenty

K ostatním komponentům patří všechny propojovací vodiče, cín apod. Tyto komponenty popisovat by bylo kontraproduktivní, proto zmíním pouze ultrazvukové senzory, které se budou starat o snížení rizika, že ať už vědomě či nevědomě, s kvadrokoptérou nabourám do případné překážky.

1.6.1 Arduino ultrazvukový měřič vzdálenosti HC-SR04

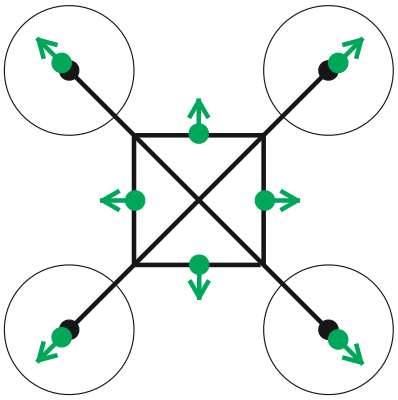
Ultrazvukový měřič vzdálenosti HC-SR04 jsem zvolil zejména z důvodu jednoduchého připojení k Arduinu Due, protože se jedná o komponentu z rodiny Arduino. Požaduje napájecí napětí 5 V a je schopný měřit vzdálenosti od 2 cm do 450 cm s udávanou přesností až 3 mm. Vysílá zvukovou vlnu o frekvenci 40 kHz. Nekomunikuje po sběrnici, ale pouze posílá analogové hodnoty, ze kterých musíme spočítat naměřenou vzdálenost. Naštěstí za tímto účelem existuje pro Arduino knihovna, která obsahuje všechny výpočty a v podstatě já jen odebírám výslednou hodnotu. Ilustrační obrázek Arduino ultrazvukového měřiče vzdálenosti HC-SR04 viz obr. 1.21. [22]

Obr. 1.21: Ultrazvukový měřič vzdálenosti HC-SR04

Pracovní úhel je pouze maximálně 15°, takže musím správně zvolit místa, kde budou umístěny. Prozatím pracuji s verzí umístění na každé rameno kvadrokoptéry a na každou stranu obdélníkové základny, tzn. bude třeba 8 senzorů – méně by podle mě bylo již málo, protože se záběrovým úhlem maximálně 15° pokryji s 8 senzory 120°, tzn. každý třetí stupeň a méně senzorů by již nepokrylo plochu dostatečně viz obr. 1.22, kde zelené tečky označují umístění ultrazvukového senzoru a šipka ukazuje smysl jejich natočení, tzn. střed snímané oblasti. Ještě bych poznamenal, že senzory budou snímat zatím pouze 2D prostor, v podstatě 2D plochu, kterou tvoří kvadrokoptéra při pohledu kopírujícím osu *roll*, tj. shora. [22]

Obecný princip ultrazvukového měřiče vzdálenosti

Nejprve se vyšle zvuková vlna o určité frekvenci a amplitudě pomocí vysílače. Vlna se následně odrazí od překážky a s určitým zpožděním závislým na rychlosti šíření vlny, se vrátí zpět a je přijata pomocí přijímače. Pomocí hodnoty zpoždění s jakým byl



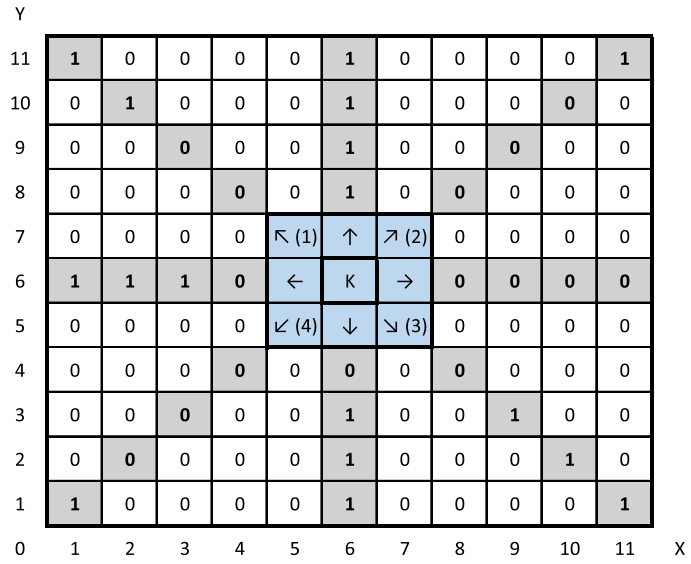
Obr. 1.22: Rozmístění ultrazvukových senzorů

vyslaný impuls přijat zpět oproti času vypuštění a znalostí rychlosti šíření vyslané vlny v určitém prostředí, vypočítám vzdálenost s jakou se nachází překážka od senzoru.

Počítačové zobrazení pomocí ultrazvukových senzorů

Způsob pro zobrazení a následné vyhodnocení, které jsem vymyslel se bude dobře demonstrovat obr. 1.23. Na obr. 1.23 najdeme kvadrokoptéru označenou písmenem *K*, kolem ní jsou šipkami označené jednotlivé ultrazvukové senzory (šipka ukazuje jejich nasměrování) a ještě čísly v závorkách označené jednotlivé motory. Nuly znamenají místo, kde není překážka a jedničky naopak místo, kde se nachází překážka.

V podstatě na začátku si experimentálně zjistím efektivní minimální a maximální dosah ultrazvukového senzoru. Pak zvolím vzdálenost, kterou bude představovat vnější tučná čára okolo kvadrokoptéry (všechno vychází z obr. 1.23) a vzdálenost která bude představovat okraj zobrazeného pole. Po-té si tyto vzdálenosti odečtu a vydělím počtem polí u každého senzoru a tím získám reálné délky jednotlivých polí viditelných na obr. 1.23 – např. jedno pole představuje reálnou délku 20 cm (pole jsou čtverce), tzn., že senzor mezi motorem (3) a (4) informuje o překážce ve vzdálenosti 20-40 cm od senzoru. Takové zobrazení reálném programování jednoduše nahradit např. dvourozměrným polem. Pak už je jen otázka nastavení citlivosti – kolik polí bude před sebou mít každý senzor.



Obr. 1.23: Návrh počítačového zobrazení 2D okolí pomocí ultrazvukových senzorů

1.7 Vývojové prostředí pro Arduino Due – Arduino IDE

Arduino IDE (*Integrated Development Enviroment*) je oficiální vývojové prostředí pro Arduino vývojové desky. Arduino IDE je napsané v jazyce java a vzniklo z výukového prostředí Processing – bylo mírně upraveno a doplněno o nezbytné funkce a k 8.11.2015 je k dispozici verze 1.6.6. Arduino dále nabízí vedle verzí pro Windows i verzi pro Mac OS nebo linux, vše jako obvykle je open-source.

Programovací jazyk pro Arduino IDE vychází z jazyků C a C++ a ještě částečně z javy. Syntaxe je ovšem trochu rozdílná. Zdrojový kód obsahuje dvě základní funkce, bez kterých by nebyla funkce programu možná:

- setup(){} – Jako první je funkce určená pro nastavení nezbytných náležitostí (přípravná) pro chod programu (různé inicializace), ale ne ve smyslu deklarací (globálních) proměnných, ale např. deklarujeme smysl použitých pinů funkci, tzn. zda budou mít vstupní nebo výstupní smysl, inicializujeme zde sběrnice a jejich parametry apod. Pozor, zde vytvořené proměnné nemají globální platnost, ale pouze platnost v této funkci! Funkce setup se provádí tedy pouze jednou a to na začátku programu.
- loop(){} – Nebo-li tzv. smyčka. Hlavní funkce, kde se provádí samotný program. Základní vlastností je, že se neustále opakuje dokola, dokud mu neřekneme ať se např. pozastaví/ukončí.

V podstatě normální funkce programu je možná už i ve funkci setup, ale není zvykem psát samotný algoritmus už tam, ale využívat setup blok jen pro inicializace

apod. Globální proměnné, import knihoven popř. vytváření objektů se provádí nad setup blokem, naopak samostatné funkce je zvykem psát až za loop. Samozřejmě je více možností jak udělat syntaxi, ale výše popsané rozvržení je zajeté a přehledné.

1.7.1 Porovnání s ostatními vývojovými prostředími

Co se týče kompilace, je Arduino IDE poněkud těžkopádné v porovnání s jinými vývojovými prostředími jako jsou např. Eclispse IDE pro javu nebo Microsoft Visual studio pro C a C++. Jednoduše řečeno, kompilace trvá poněkud dlouho a časté odlaďování je více než nepříjemné a časově náročné. To ještě zhoršuje absence klasického debug módu na jaký jsem byl zvyklý z Eclipse IDE nebo Microsoft Visual Studia, kde jsou přehledně vypsány použité proměnné. Kdežto u prostředí Arduino IDE takový typický debug mód neexistuje a musíme debugovat tím způsobem, že do samotného kódu na místa, kde chceme číst určité proměnné, napsat např. příkaz *Serial.print(x)*, který nám do sériové linky následně vypíše požadovanou informaci obsaženou v proměnné x. Prostředí Arduino IDE nám totiž umožňuje virtuální prohlížení sériových linek s různými modulačními rychlostmi v baudech (značka *Bd*).

2 PRAKTICKÉ VÝSLEDKY PRÁCE

2.1 Finální realizace

Fyzická realizace a zapojení komponent zůstalo stejné, jak jsem popisoval v kapitole 1.1.3, respektive viz obr. č. 1.7, až na dvě výjimky. Po několika praktických testech v reálných podmínkách jsme s Panem doktorem Krajsou dospěli k názoru, že rozmístění os bude pro reálnou aplikaci nejvíce praktické dle metody 2 na obr. 1.4 a to z důvodu logického (programového) rozmístění programových PID. Tzn. osy gyroskopu budou kopírovat ramena kvadrokoptéry, takže na stabilizaci každé osy mi stačí jeden virtuální PID regulátor – vyberu z každé ze dvou vodorovných os jeden motor, který bude kontrolovat náklon v dané ose, zatímco druhý motor bude mít statický výkon, který se bude pouze zvedat při pokynu z ovladače (např. pro let vzhůru). Z toho vyplývá, že budu mít pro dvě vodorovné osy dva virtuální PID regulátory, které budou každý na jednom motoru v dané ose, takže další dva zbylé motory budou bez zásahu PID – jeden z nich jsem následně vybral a určil ho pro regulaci třetí (svislé) osy, tzn. bude hlídat, aby se kvadrokoptéra neotáčela kolem vlastní osy (po nebo proti směru hodinových ručiček), ale aby stabilně držela takové natočení, jaké je po ní požadováno přes vysílač (svislá osa).

A druhá změna nastala ve způsobu ovládání ESC. V kapitole 1.1.3 popisuji, že ESC budou řízeny PPM signálem (modulací) s odstupem mezi jednotlivými pulsy 50 ms, kde informaci nese druhá polovina délky pulsu – puls je min. dlouhý 1 ms a max. 2 ms. Zatímco později jsem zjistil, že ESC lze řídit i PWM modulací o kmitočtu 490 Hz, kde informaci nese střída – 50% je min. výkon (vypnuto nebo jinak řečeno, nulové otáčky) a 100% je max. výkon (maximální otáčky).

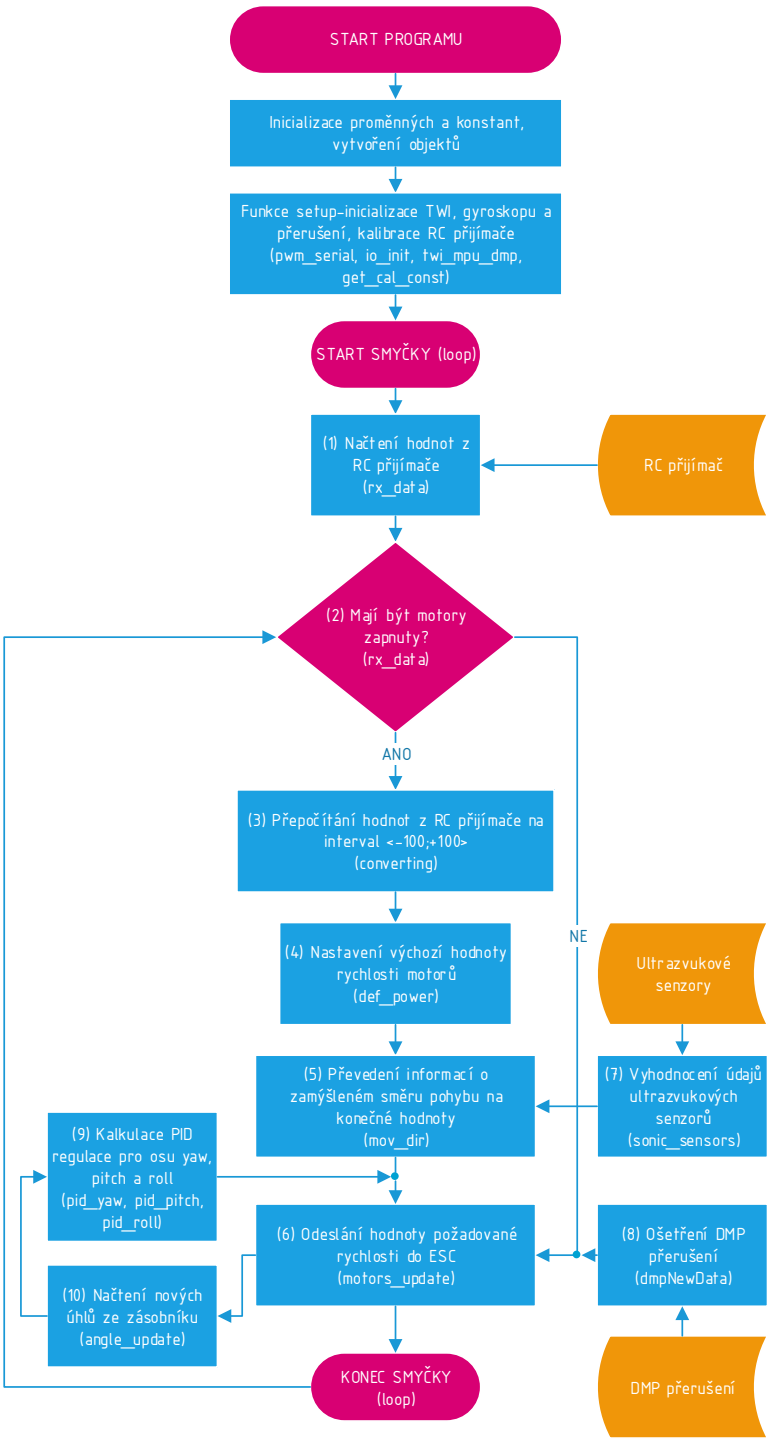
Při samotném sestavení vyskytlo několik menších problémů. Nejprve to byl problém s malým průřezem silových vodivých cest zabudovaných uvnitř rámu, které rozvádí elektrický proud k ESC. Při mém zátěžovém testu nevydržely ani 5 s zatížení 32 A, přičemž by měl vydržet alespoň 40 A (myšleno 32 A respektive 40 A pro všechny 4 ESC dohromady). Takže jsem musel toto místo přemostit měděným vodičem průřezu 2,5 mm^2, běžně používaným v domovních rozvodech při 16 A jističích. Druhá větší komplikace byla, upevnit gyroskop tak, aby vodorovné osy gyroskopu kopírovaly vodorovné osy kvadrokoptéry. Což se mi povedlo s přesností $\pm 1°$. Ostatní sestavování šlo více méně bez problémů, takže jsem se dále soustředil na samotný řídící kód. Pozn.: umístěná 8 ultrazvukových senzorů by bylo kontraproduktivní (jak popisuji na obr. 1.23 v kapitole 1.6.1), proto jsou umístěny jen 4 ve vodorovné poloze, vzájemně otočených o 90°. Ještě jsem jeden senzor přidal aby hlídal náraz např. do stropu při stoupání.

2.2 Vývojový diagram

Na obr. 2.1 lze vidět vývojový diagram kódu. Nyní stručně popíšu na základě obr. 2.1 celý program jako takový z celkového blokového pohledu. Ovšem v následující kapitole 2.3 popíšu funkci jednotlivých bloků (částí/funkcí) podrobně. Názvy v závorce na konci každého bloku značí, jak se každý blok (funkce) reálně jmenuje v kódu.

Úvodní blok *START PROGRAMU* značí moment, kdy je Arduino připojeno k napájení, je tedy zapnuto. Následuje nejprve zahrnutí hlavičkových souborů (knihoven), dále inicializace globálních proměnných a konstant a vytvoření nezbytných objektů. Následuje funkce `setup`, kde proběhne hlavně inicializace periferií. Tzn. TWI sběrnice pro komunikaci s gyrem, určení, které piny budou vstupní a které výstupní, připnutí přerušení k pinům a jejich způsob ošetření a na co budou reagovat (nástupná, sestupná...). Nakonec funkce `setup` se provede jednoduchá kalibrace RC přijímače.

Následuje samotná nekonečná smyčka běhu programu (*START SMYČKY*). Pro přesnost na začátek uvedu, že kdykoli v průběhu programu může MPU6050 vyvolat přerušení, takže průběh programu skočí do bloku (8). Jako první tedy načteme příchozí hodnoty z RC přijímače a následně dojde k rozhodnutí, zda-li mají být motory zapnuty (a má probíhat většina výpočtů, smyčka pokračuje po ANO větvi dál) nebo vypnuty (program skočí do NE větve a motory se v bloku (6) vypnou). Motory tedy mají být zapnuty tak pokračujeme do bloku (3), kde přepočítám údaje z RC přijímače na hodnoty, se kterými se bude snáze počítat (interval $<-100;+100>$). Pak nastavím výchozí hodnotu rychlosti motorů, se kterou probíhají výpočty (4). Po-té údaje, které jsem získal převedením hodnot z RC přijímače, znova převedu na hodnoty, které již vstupují do PID jako požadovaná hodnota – blok (5) – a do finálního přepočtu (6). Tyto hodnoty jsou nejprve ještě na začátku tohoto bloku (5) korigovány vyhodnocenými hodnotami z ultrazvukových senzorů (7). Nyní tedy postoupím dál do (6), kde se spočítá finální rychlost, ale pouze tehdy pokud jsou k dispozici nové hodnoty náklonu, tzn. přišlo přerušení z gyra (8). Předpokládám, že nyní přišlo, tak hodnoty vyčtu ze zásobníku a přepočítám na stupně (10) a následně pošlu do (9), kde proběhne kalkulace regulace na hodnotu, kterou požaduji. Pak se znovu vracím do (6) a vyšlu nové údaje směrem k ESC (respektive motorům). Což byl poslední krok a nyní se smyčka přesouvá znova k načtení nových hodnot z RC přijímače a to se děje i v případě, když byl průběh po NE větvi.



Obr. 2.1: Vývojový diagram

2.3 Řídící kód

V této kapitole popíšu jednotlivé části řídícího kódu, tak jak jdou po sobě. Nejprve v části nastavování – funkce `setup` – a po-té v řídící smyčce (tzv. `loop`). Vynechám deklarace proměnných, konstant a vytváření objektů a podobné režijní operace, pouze nejpodstatnější proměnné, konstanty a objekty shrnu do tří tabulek, viz tab. pro konstanty 2.1, tab. pro proměnné 2.3 a tab. pro objekty 2.2. Následně ještě uvedu použité hlavičkové soubory (knihovny) – viz tab. 2.4.

Tab. 2.1: Důležité konstanty

Konstanta	**Stručný popis**
`cal_const`	Kalibrační konstanta upravující hodnoty přijímané Arduinem z přijímače.

`YAW_P/I/D_VAL`
`PITCH_P/I/D_VAL`
`ROLL_P/I/D_VAL`
	Regulační konstanty pro každý virtuální PID regulátor.

`YAW_OUTPUT_MIN/MAX`
`PITCH_OUTPUT_MIN/MAX`
`ROLL_OUTPUT_MIN/MAX`
	Maximální rozmezí, ve kterém může virtuální PID regulátor regulovat.

`default_power`	Experimentálně zjištěná hodnota velikosti rychlosti jdoucí do ESC za účelem ovládání motorů, kdy jsou motory zhruba na 80% otáček potřebných pro vznášení se.
`middle_low` `middle_high`	Experimentálně změřené hodnoty, ve kterých se pohybuje délka PPM pulsu přijímaných Arduinem při klidové (nulové) poloze pák vysílače.

Tab. 2.2: Použité objekty

Jméno objektu	Stručný popis
motor1 motor2 motor3 motor4	Objekty pro ovládání každého jednotlivého motoru na základě knihovny Motor32.
mpu	Objekt pro komunikaci s čipem MPU6050 na základě knihovny I2Cdev.

Tab. 2.3: Důležité proměnné

Proměnná (typ)	**Stručný popis**
`finalPower[4]` (int)	Konečná informace o požadované velikosti rychlosti jdoucí přes ESC do motorů (už i s dynamickou (PID) složkou). V podstatě je to hodnota délky doby HIGH úrovně v mikrosekundách v 490 Hz PWM.
`power[4]` (int)	Tzv. offsetový základ pro `finalPower[4]`, jde o hodnotu odvozenou od konstanty `default_power`.
`rx[5]` (int)	Velikost doby trvání pulsu na každém z pěti přijímaných kanálů (PPM modulace) v mikrosekundách, tak jak je přijímač posílá do Arduina.
`dir[4]` (int)	Překonvertovaná hodnota z prvních čtyř hodnot z pole `rx[5]` na hodnotu ±100, která určuje směr, který skrz vysílač požaduji – kladný hodnota dopředu/doprava/nahoru/natáčení po směru hodinových ručiček a záporná hodnota analogicky opačné směry; při nulové hodnotě jsou páky vysílače ve výchozí poloze.
`wanted_yaw` `wanted_pitch` `wanted_roll` (double)	Požadovaná hodnota natočení v osách *yaw* (svislá osa) a *pitch* a *roll* (vodorovné osy), které vstupují do PID jako požadovaná hodnota (tzv. *setpoint*).
`realYPR[3]` (double)	Hodnota velikosti úhlu pro každou osu ve stupních vstupující do PID.

Tab. 2.4: Připojené hlavičkové soubory

Jméno hlavičkového souboru	Stručný popis
Wire	Knihovna pro komunikaci po sériové lince, je nezbytná pro správnou funkci knihovny I2Cdev.
I2Cdev	V podstatě jediná knihovna dotažená do konce, zajišťující komunikaci po I2C respektive TWI sběrnici.
MPU6050_6Axis_MotionApps20	Komunikace s MPU6050 (DMP).
Motor32	Vytváření PWM na 32 bitové ARM architektuře.

2.3.1 Funkce `setup`

Zde popíšu jednotlivé bloky nastavovací funkce `setup`, která je provedena po zahrnutí knihoven, deklaraci proměnných, konstant a vytvoření objektů. Pozn.: Klíčové slova jsou barevně odlišeny.

Inicializace PWM, ESC a rychlosti sériové linky

Nejprve inicializuji pomocí funkce `pwm32_init` používání PWM pomocí knihovny Motor32. Jinak řečeno, drtivá většina Arduin je založena na AVR architektuře (8 bitová či 16 bitová), takže standardní způsoby vytváření PWM fungující na AVR pomocí Arduino funkcí, již nefungují nevyzpytatelně na 32 bitové ARM architektuře (Arduino DUE). Hlavní problém je v použití časovačů, takže výsledný průběh bez použití knihovny Motor32 vykazuje náhodně proměnou střídu a náhodné zákmity. Řídit ESC bez použití Motor32 je prakticky i teoreticky nemožné. Takže po dlouhém hledání jsem našel právě knihovnu Motor32, která byla vytvořena právě k účelu generování PWM a PPM na ARM 32 bitové architektuře o kmitočtu určeném k řízení ESC. Pozn.: pokud tyto Arduino funkce pro generování PPM a PWM použiji samostatně (mimo můj kód), tak fungují správně.

Jak lze vidět dále objekty `motor1` až `motor4` používají metodu `set`, do které vstupuje celé kladné číslo (např. typu unsigned integer). Číslo vstupující je v reálu šířka pulsu ve 490 Hz PWM psaná v mikrosekundách. Hodnota 1000 μs je zde proto, že ESC vyžadují pro své počáteční nastavení minimální hodnotu plynu, což je právě 1000 μs (max. hodnota jak jsem již uvedl je 2000 μs). Nakonec na posledním řádku odstartuji sériovou linku, kde číslo značí rychlost komunikace v baudech. V kódu tento blok vystupuje jako funkce `pwm_serial`.

```
1 void pwm_serial() {
2  pwm32_init();
3  motor1.set(1000);
4  motor2.set(1000);
5  motor3.set(1000);
6  motor4.set(1000);
7  Serial.begin(115200);
8 }
```

Inicializace vstupů a výstupů z Arduina

Zde jednoduše řekneme Arduinu pomocí standardních Arduino funkcí, které piny budou vstupy a které výstupy. Je jich méně než by se dalo očekávat, ale ostatní vstupy/výstupy jsou již deklarovány v rámci knihoven, které je používají. Tzn. zde jsou inicializovány pouze vstupy z RC přijímače (2. až 5. řádek) a vstupy a výstupy ultrazvukových senzorů (6. až 10. řádek). V kódu jako funkce `io_init`.

```
1 void io_init() {
2  for(int i=0;i<4;i++) {
3   pinMode(30+i, INPUT);
4  }
5  pinMode(35, INPUT);
6  pinMode(TRIGPIN_0, OUTPUT);pinMode(ECHOPIN_0, INPUT);
7  pinMode(TRIGPIN_1, OUTPUT);pinMode(ECHOPIN_1, INPUT);
8  pinMode(TRIGPIN_2, OUTPUT);pinMode(ECHOPIN_2, INPUT);
9  pinMode(TRIGPIN_3, OUTPUT);pinMode(ECHOPIN_3, INPUT);
10 pinMode(TRIGPIN_4, OUTPUT);pinMode(ECHOPIN_4, INPUT);
11 }
```

Inicializace TWI sběrnice, MPU6050 a DMP

Na druhém a třetím řádku se připojíme na TWI sběrnici a pak inicializujeme MPU6050. Na 4. až 14. řádku se pokoušíme inicializovat DMP. Nejprve se přesvědčíme, zda vůbec lze komunikovat s DMP – rozhodovací `if`. Pokud lze komunikovat, tak povolíme používání DMP (5. řádek), povolíme externí přerušení na pinu 22 reagující na nástupní hranu a skokem do funkce `dmpDataReady`. A nakonec zjistíme, jak velké pakety DMP vrací (10. řádek). Pokud se nepovede připojit k DMP tak se program zastaví a musíme vše restartovat. Nakonec tohoto bloku vystupujícího pod jménem `twi_mpu_dmp`, je na posledních dvou řádcích nastavení citlivost měření náklonu a akcelerace – citlivost lze nastavovat jen po konkrétních hodnotách, proto jsou použity konstanty, které jsou deklarované v knihovně MPU6050_6Axis_MotionApps20.

```
1 void twi_mpu_dmp() {
2  Wire.begin();
3  mpu.initialize();
4  if (mpu.dmpInitialize() == 0) {
5   mpu.setDMPEnabled(true);
6   attachInterrupt(22, dmpNewData, RISING);
7   mpuIntStatus = mpu.getIntStatus();
8   dmpReady = true;
9   packetSize = mpu.dmpGetFIFOPacketSize();
10 }
11 else {
12  delay(60000);
13 }
14 mpu.setFullScaleGyroRange(MPU6050_GYRO_FS_2000);
15 mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
16 }
```

Získání kalibrační konstanty pro RC přijímač

Jako poslední blok funkce `setup` získáme kalibrační konstantu jednoduše zjištěním průměrné střední hodnoty PPM signálu z RC přijímače, kterou odečteme od teoretické střední hodnoty (1505 μs). Bližší popis funkce `pulseIn` a důvodu zakázaní přerušení na 4. řádku, viz následující podkapitola. Tento blok vystupuje v kódu jako `get_cal_const`.

```
1 void get_cal_const(){
2  unsigned int x=0;
3  for(int i=0;i<4;i++) {
4   noInterrupts();
5   x=x+pulseIn(i+30, HIGH);
6   interrupts();
7  }
8  cal_const=1505-x/4;
9 }
```

2.3.2 Funkce `loop`

Načtení informací z RC přijímače do Arduina Due

Jako první se ve smyčce změří velikosti doby trvání příchozích PPM pulsů z RC přijímače, které by měli teoreticky být v rozmezí 1 ms – 2 ms, ovšem reálně se odchylka pohybuje v rozmezí $\pm 100\,\mu$s. Takže cyklem `for` projedu první čtyři přijímané kanály (fyzicky páky na vysílači). Pomocí Arduino funkce *pulseIn*, u které první parametr znamená pin, kde se má příchozí puls měřit a druhý na jakou hodnotu má funkce reagovat. Tzn. analogicky by jsem mohl zadat i `LOW` a funkce by reagovala na příchozí logickou nulu a měřila by po jakou dobu se tam nachází. Následně se hned po změření přičte kalibrační konstanta (`cal_const`) získaná před započetím první smyčky, tj. ve funkci `setup`. Po načtení prvních čtyř kanálů se nakonec načte i pátý kanál (6. řádek), který je představován fyzicky pouze dvoustavovým přepínačem a určil jsem ho jako vypnutí/zapnutí motorů – proto se hned na následujícím řádku zkouší jestli není v poloze vypnuto a pokud ano, tak program neprodleně skočí do funkce `motors_update`, kde se ihned vyšle pokyn k vypnutí motorů. Při Arduino funkci `pulseIn` je bezpodmínečně nutné zakázat přerušení pomocí Arduino funkce `noInterrupts` (2. řádek), jinak funkce `pulseIn` nebude fungovat. Nakonec opět přerušení povolíme (8. řádek).

```
1 void rx_data() {
2  noInterrupts();
3  for(int i=0;i<4;i++) {
4   rx[i]=pulseIn(i+30, HIGH)+cal_const;
5  }
6  rx[4]=pulseIn(35, HIGH);
7  if(rx[4]<1700) {motors_update();}
8  interrupts();
9 }
```

Převedení informací z RC přijímače

Po načtení délek pulsů, si tyto hodnoty převedu do rozmezí ± 100, aby se s nimi lépe pracovalo. Ale to vše jen pro čtyři pákové kanály – pátý nechám tak, protože je pouze dvoustavový, kdy v hodnotě zapnuto má puls délku přes 1800 μs a při vypnuto má pod 1200 μs, takže hodnoty jsou jednoznačné.

Cyklem `for` projedu všechny čtyři kanály a pro každý nejprve zjišťuji, jestli se nachází páka ve středu – tj. v rozmezí ohraničeném experimentálně zjištěnými konstantami `middle_low` a `middle_high` a případně přiřadím pro daný směr (direction – `dir`) nulu. V opačném případě se na základě konkrétního kanálu nejprve rozhoduji zda-li není hodnota vyšší respektive menší než experimentálně zjištěná maximální respektive minimální hranice. Pokud ani to, tak dále namapuji danou hodnotu do rozmezí ±99. A tak analogicky pro každý kanál. Tzn. nyní se naplnilo pole hodnot `dir[4]` hodnotami ±100, se kterými se bude dále lépe pracovat než se surovými daty z `rx[5]`.

Pracuji zatím jen experimentálně ještě s možností odstranění kalibrační konstanty a kalibrací na konkrétní max. a min. hodnotu a střední interval ve funkci *setup*. Prozatím si ale vystačím s již uvedeným popisem, který ±10% odpovídá.

```
1  void converting() {
2   for(int i=0;i<4;i++) {
3    if(middle_low<rx[i] && rx[i]<middle_high) {dir[i]=0;}
4    else {
5     switch(i) {
6      case 0:
7       if(rx[i]>1960) {dir[i]=100;}
8       if(rx[i]<1130) {dir[i]=-100;}
9       if(rx[i]<=middle_low && rx[i]>=1130) {dir[i]=map(rx[i],1130,middle_low
            ,-99,-1);}
10      if(rx[i]>=middle_high && rx[i]<=1950) {dir[i]=map(rx[i],middle_high
            ,1950,1,99);}
11     break;
12     case 1:
13      if(rx[i]>1840) {dir[i]=100;}
14      if(rx[i]<1170) {dir[i]=-100;}
15      if(rx[i]<=middle_low && rx[i]>=1170) {dir[i]=map(rx[i],1170,middle_low
            ,-99,-1);}
16      if(rx[i]>=middle_high && rx[i]<=1840) {dir[i]=map(rx[i],middle_high
            ,1840,1,99);}
17     break;
18     case 2:
19      if(rx[i]>1950) {dir[i]=100;}
20      if(rx[i]<1200) {dir[i]=-100;}
21      if(rx[i]<=middle_low && rx[i]>=1200) {dir[i]=map(rx[i],1200,middle_low
            ,-99,-1);}
22      if(rx[i]>=middle_high && rx[i]<=1950) {dir[i]=map(rx[i],middle_high
            ,1950,1,99);}
23     break;
24     case 3:
25      if(rx[i]>1850) {dir[i]=100;}
26      if(rx[i]<1165) {dir[i]=-100;}
27      if(rx[i]<=middle_low && rx[i]>=1165) {dir[i]=map(rx[i],1165,middle_low
            ,-99,-1);}
28      if(rx[i]>=middle_high && rx[i]<=1850) {dir[i]=map(rx[i],middle_high
            ,1850,1,99);}
29     break;
30    }
31   }
32  }
33 }
```

Nastavení výchozí hodnoty pro rychlost motorů

Následně nahraji do průběžné proměnné `power` offsetovou hodnotu pro rychlost motorů, která se bude v dalším kroku zvyšovat/snižovat v závislosti na požadavku letu nahoru/dolů. Tento systém získávání konečné rychlosti by se dal samozřejmě zjednodušit do méně kroků, ale volím tento způsob pro přehlednější práci a popis kódu. V kódu jako funkce `def_power`.

```
1 void def_power() {
2  for(int i=0;i<4;i++) {
3   power[i]=default_power;
4  }
5 }
```

Získání informací o požadovaném směru pohybu

Nyní převedu žádaný pohyb v určitém směru (`dir[5]`) na reálně žádané náklony v osách pro PID regulátory nebo reálně žádané výkony za účelem stoupání/klesání (offset). Experimentálně jsem musel zjistit v jakém směru bude na ose buď kladný nebo záporný náklon a podle toho jsem, jak lze vidět níž, namapoval[2] rozmezí ±100 na rozmezí požadovaného náklonu ±20° – tento náklon jsem zadal z důvodu hladkého testování. Samozřejmě pro agresivnější let by se zadal větší náklon. Toto tedy platí pro `dir[0]` (let vpřed/vzad) a `dir[1]` (let vpravo/vlevo). `dir[2]` je směr klesání/stoupání – mapuje se zde o kolik má být zvednuta/snížena hodnota oproti `default_power`. Nyní je nastaveno 20%, ale lze opět tuto hodnotu zvýšit pro agresivnější stoupání. Nakonec `dir[3]` pouze určuje jeli páka vpravo/vlevo a zda-li se má konstantní rychlostí (0.01° za jeden průběh smyčky) začít otáčet kolem svislé osy (*yaw*) po nebo proti směru hodinových ručiček. Ano je zbytečné zde mít rozmezí ±100, ale je to příprava pro případné budoucí úpravy kódu pro agresivnější let, kdy mi nebude stačit otáčení kolem svislé osy konstantní rychlostí. V kódu pod označením `mov_dir`. O funkci `sonic_sensors` (2. řádek) více v následující podkapitole.

```
1 void mov_dir() {
2  sonic_sensors();
3 //Let vpred/vzad
4  if(dir[0]==0) {wanted_roll=0;}
5  else {
6   if(dir[0]>1) {wanted_roll=map(dir[0], 1, 100, 0, 200000)/10000.000;}
7   if(dir[0]<0) {wanted_roll=map(dir[0], -100, 1, -200000, 0)/10000.000;}
8  }
9 //Let doprava/doleva
10  if(dir[1]==0) {wanted_pitch=0;}
11  else {
12   if(dir[1]>0) {wanted_pitch=map(dir[1], -100, 1, 200000, 0)/10000.000;}
13   if(dir[1]<0) {wanted_pitch=map(dir[1], 1, 100, 0, -200000)/10000.000;}
14  }
```

[2]Pozn.: počítání s tak vysokými hodnotami jsem musel zvolit proto, že funkce `map` umí pracovat pouze s datovými typy integer.

```
15 //Let nahoru/dolu
16  if(dir[2]>0) {
17   power[0]=power[0]*(map(dir[2], 1, 100, 10000, 12000)/10000.000);
18   power[1]=power[0];power[2]=power[0];power[3]=power[0];
19  }
20  if(dir[2]<0) {
21   power[0]=power[0]*(map(dir[2], -100, -1, 9000, 10000)/10000.000);
22   power[1]=power[0];power[2]=power[0];power[3]=power[0];
23  }
24 //Natoceni kolem vlastni osy po nebo proti smeru hodinovych rucicek
25   if(dir[3]>0) {wanted_yaw=wanted_yaw+0.01;}
26   if(dir[3]<0) {wanted_yaw=wanted_yaw-0.01;}
27 }
```

Korekce směru pohybu na základě ultrazvukových senzorů

Nyní se dostávají ke slovu ultrazvukové senzory, které hlídají, či lépe řečeno snižují šanci srážky s překážkou. Na řádku kódu 2. až 9. se vyšle pokyn do senzoru k vyslání zvukového pulsu ze senzoru, který se případně odrazí od blízké překážky (echo) a senzor případný odraz zachytí. Následně pak senzor případný odraz zpracuje a odešle odpověď do Arduina v podobě určité délky pulsu (doba po kterou je logická 1 – `HIGH`), kde délka pulsu nese informaci o vzdálenosti překážky od senzoru. Na řádku 8 se tedy puls změří a pomocí konstanty převede na jednotky délky (zde na centimetry). A uloží do proměnné `dist_cm[i]`, která je pole typu float. Každý jeden prvek pole odpovídá jednomu fyzickému senzoru.

Ve zbylé části vyhodnocuji zda má kvadrokoptéra přikázáno pohyb určitým směrem (proměnná `dir[i]`) a zároveň zda není méně jak 70 cm od překážky. Potom se proměnná `dir[i]` nastaví na 0, tzn. natvrdo zamezíme pohybu tím směrem. Dále jsem přidal ochranu proti nebezpečnému přiblížení na méně jak 50 cm. Pokud dojde k nebezpečnému přiblížení, program nastaví pohyb v opačném směru ve 30 % maximální rychlosti. Pozn.: max. rychlost je určována náklonem, kde nyní mám nastaveno 20° pro vodorovné osy – viz předchozí podkapitola. Za tímto účelem jsou tedy senzory 0 až 3 (obdobně 0. až 3. prvek pole `dir[i]`). Čtvrtý senzor hlídá nebezpečné stoupání např. ke stropu. Způsob ohlídání přiblížení je na stejném principu, jak hlídání vodorovných os, pouze jsem poupravil konstanty – namísto -30 % jsem dal -10 %, aby motory neztratily zásadní tah.

Tato realizace není přesně podle toho jak jsem nastiňoval v kap. 1.6.1. Podle toho by totiž byla realizace zbytečně složitější, jak se v praxi později ukázalo.

```
1 void sonic_sensors() {
2  for(int i=0;i<5;i++) {
3   digitalWrite(44+i, LOW);
4   delayMicroseconds(2);
5   digitalWrite(44+i, HIGH);
6   delayMicroseconds(10);
7   digitalWrite(44+i, LOW);
8   dist_cm[i]=pulseIn(49+i, HIGH)*0.017315f;
9  }
```

```
10  if(dir[0]>0 && dist_cm[0]<70) {
11   dir[0]=0;
12   if(dist_cm[0]<50) {dir[0]=-30;}
13  }
14  if(dir[0]<0 && dist_cm[2]<70) {
15   dir[0]=0;
16   if(dist_cm[2]<50) {dir[0]=30;}
17  }
18  if(dir[1]>0 && dist_cm[3]<70) {
19   dir[1]=0;
20   if(dist_cm[3]<50) {dir[1]=-30;}
21  }
22  if(dir[1]<0 && dist_cm[1]<70) {
23   dir[1]=0;
24   if(dist_cm[1]<50) {dir[1]=30;}
25  }
26  if(dir[2]>0 && dist_cm[4]<70) {
27   dir[2]=0;
28   if(dist_cm[4]<50) {dir[2]=-10;}
29  }
30 }
```

Vyslání výsledné požadované hodnoty rychlosti motorů do ESC

V bloku jménem `motors_update` se vyšle výsledná hodnota požadované rychlosti motorů do ESC. Na řádcích 5 až 8 se vypočítají výsledné hodnoty z tzv. offsetové hodnoty rychlosti (`power[i]`), ke které se přičte hodnota PID regulace pro konkrétní motor. K tomuto dojde pouze v případě, pokud máme dvoustavový přepínač na vysílači v poloze zapnuto (2. řádek) a pokud DMP dodalo nové hodnoty (3. řádek). Pokud tedy obě podmínky platí, tak se nejprve zakáže přerušení, jelikož se jedná o časově kritickou operaci. Průběh program skočí do funkce `angle_update`, o které více v podkapitole níže. Na 9. až 12. řádku je vyslána tedy požadovaná hodnota směrem k ESC.

Pokud je dvoustavový přepínač v poloze pro vypnuté motory, pokračuje se na 17. řádku, kdy je nejprve opět zakázáno přerušení a po-té jsou vyslány k ESC minimální hodnoty otáček. Na 24. řádku ještě vyberu nashromážděné hodnoty z FIFO zásobníku, které přišli z DMP. Sice k jsou ničemu, ale je třeba udržovat FIFO co nejméně zaplněné.

Ještě zmíním funkci `dmpNewData`, která je volána když DMP vyšle přerušení a proměnná `mpuInterrupt` se nastaví na `true`. To značí, že jsou nachystány nové data o náklonu (DMP). Tento úkon jsem zařadil z důvodu, aby se do ESC nevysílaly hodnoty, které se již vysílaly (3. řádek funkce `motors_update`. A též abych ošetřil časté příchody přerušení od DMP, tzn. aby CPU neustále neodbíhal ke složité obsluze přerušení (výběr dat z FIFO zásobníku).

```
1 void dmpNewData() {
2  mpuInterrupt = true;
3 }
```

```
void motors_update() {
 if(rx[4]>1700) {
  if(mpuInterrupt) {
   noInterrupts();
   angle_update();
   finalPower[0]= (int) power[0];
   finalPower[1]= (int) power[1]+pid_yaw(realYPR[0], wanted_yaw);
   finalPower[2]= (int) power[2]+pid_roll(realYPR[2], wanted_roll);
   finalPower[3]= (int) power[3]+pid_pitch(realYPR[1], wanted_pitch);
   motor1.set(finalPower[0]);
   motor2.set(finalPower[1]);
   motor3.set(finalPower[2]);
   motor4.set(finalPower[3]);
   interrupts();
  }
 }
 else {
  noInterrupts();
  motor1.set(1000);
  motor2.set(1000);
  motor3.set(1000);
  motor4.set(1000);
  interrupts();
  if(mpuInterrupt) {angle_update();}
 }
}
```

PID regulátor

V praxi se ukázalo, že je nutné PID navrhnout sofistikovanějšími metodami. Já jsem zvolil jednoduchou variantu. Protože návrh odpovídajícího PID (respektive návrh stabilizace) je úkon, u něhož se předpokládají předchozí zkušenosti s návrhem. Šlo by o vypracování modelu regulované soustavy (kvadrokoptéra), tj. modelování chování odezvy motorů (akčních členů) např. na jednotkový skok apod. Následně bych tedy zvolil určitý typ PID, vytvořil přenosovou rovnici a dopočítal regulační konstanty. A to by mohl být možný námět případné diplomové práce.

Níže uvedený PID regulátor je pro osu *roll*, ale pro zbylé dvě osy se mění pouze regulační konstanty (`ROLL_P_VAL...`) a meze, mezi kterými může probíhat regulace (`ROLL_OUTPUT_MAX...`). Takže jsem uvedl jen tento příklad, takže jméno příslušného PID je buď `pid_yaw`, `pid_pitch` nebo `pid_roll`.

```
double pid_roll(double real_angle, double wanted_angle){
 double input=real_angle;
 double error=wanted_angle-input;
 ITerm_roll+=(ROLL_I_VAL*error);
 if(ITerm_roll > ROLL_OUTPUT_MAX) {ITerm_roll=ROLL_OUTPUT_MAX;}
 else if(ITerm_roll < ROLL_OUTPUT_MIN) {ITerm_roll= ROLL_OUTPUT_MIN;}
 double dInput=input-lastInput_roll;
 double output=ROLL_P_VAL*error+ITerm_roll-ROLL_D_VAL*dInput;
 if(output > ROLL_OUTPUT_MAX) {output=ROLL_OUTPUT_MAX;}
 else if(output < ROLL_OUTPUT_MIN) {output = ROLL_OUTPUT_MIN;}
 lastInput_roll=input;
 return output;
}
```

Načtení nových hodnot z DMP

Pokud přišlo přerušení od DMP (MPU6050), nastaví se dvoustavová proměnná `mpuInterrupt` na hodnotu **true**, což značí, že jsou připravené nové hodnoty z DMP – ošetříme přerušení pomocí `dmpNewData` – řešení ošetření přerušení pomocí ní má velkou výhodu. Vždy totiž, když přijde přerušení, tak se obslouží jen touto z výpočtového hlediska časově nenáročnou funkcí. Na začátku vyčítání hodnot z FIFO zásobníku tedy proměnou `mpuInterrupt` uvolním (3. řádek). Na následujícím řádku získám délku obsahu FIFO zásobníku. Po-té se rozhodnu zda nedošlo k přetečení zásobníku (5. až 7. řádek), pokud ano, vyčistím FIFO. Toto opatření je kvůli tomu, aby byly příchozí pakety ve FIFO frontě v celku – v situaci, kdy by přišel paket, který se jen z půlky vleze do zásobníku, tak by se druhá půlka zahodila. Pokud tedy nedošlo k přetečení zásobníku, přikročíme k vyčtení prvního paketu, který je na řadě (8. až 11. řádek). Nejprve musím zjistit celkovou délku dat v zásobníku a pak na základě znalosti velikosti jednoho paketu vyjmu ze zásobníku data o té velikosti, které jsou první na řadě. Pak již jen pomocí metod na řádcích 12. až 14. (tyto metody jsou součástí knihovny MPU6050_6Axis_MotionApps20) extrahuji přijaté hodnoty z paketu. Po-té na 15. až 17. řádku naplním finální proměnou `realYPR` (pole typu double) výslednými úhly ve stupních. Zde je též proveden přepočet z radiánů na stupně. Na následujícím řádku vymažu celou FIFO frontu.

Mohlo se tedy stát, že jsem vymazal ještě nevyčtená data. To je z toho důvodu, že se ve FIFO frontě hromadily pakety a když se tedy průběh programu dostal k jejich vyčtení, byly aktuální hodnoty až několikáté v pořadí a CPU musel vyčítat nejprve staré nahromaděné hodnoty. Tím pádem vznikalo zpoždění takové, se kterým bylo obtížné provést i částečnou stabilizaci. Další snížení prodlevy jsem docílil zrychlením TWI sběrnice. V knihovně I2Cdev je výchozí rychlost sběrnice nastavena na poloviční hodnotu než si může Arduino dovolit komunikovat s MPU6050 – bylo nastaveno na 100 kHz a lze nastavit na 400 kHz, ale vzhledem k rychlosti snímání vzorků MPU6050 (8 kHz pro gyro a 1 kHz pro akcelerometr) a velikosti posílaných paketů, stačí rychlost zhruba 290 kHz (teoreticky), prakticky je toto číslo menší. Takže jsem sběrnici nastavil na 233 kHz. Tyto dvě úpravy vedly ke zmenšení zpoždění při získávání aktuálních hodnot náklonu z cca 300 ms na cca 50 ms.

V poslední části (řádek 19. až 22.) redukuji vliv nežádoucích náhodných chyb přijímaných hodnot. Jelikož po určité době pozorování jsem zjistil, že můžou v náhodných okamžicích přijít naprosto chybné hodnoty (např. i 100° odchýlené, ale i jen 30°). Takže jsem zařadil tuto část, kdy předpokládám, že se kvadrokoptéra nenakloní o více než ±60° ve vodorovných osách. Tomuto číslu odpovídá i charakter celého návrhu, jelikož nenavrhuji kvadrokoptéru např. pro akrobační let, ale pro jednoduchý a klidný let.

```
void angle_update() {
 mpuIntStatus = mpu.getIntStatus();
 mpuInterrupt = false;
 fifoCount = mpu.getFIFOCount();
 if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
  mpu.resetFIFO();
 }
 else if (mpuIntStatus & 0x02) {
  while (fifoCount < packetSize) {fifoCount = mpu.getFIFOCount();}
  mpu.getFIFOBytes(fifoBuffer, packetSize);
  fifoCount -= packetSize;
  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.dmpGetGravity(&gravity, &q);
  mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
  realYPR[0]=(ypr[0]*(180/M_PI));
  realYPR[1]=(ypr[1]*(180/M_PI));
  realYPR[2]=(ypr[2]*(180/M_PI));
  mpu.resetFIFO();
  if((realYPR[2]>60) || (realYPR[2]<-60)) {realYPR[2]=lastRealYPR[2];}
  else {lastRealYPR[2]=realYPR[2];}
  if((realYPR[1]>60) || (realYPR[1]<-60)) {realYPR[1]=lastRealYPR[1];}
  else {lastRealYPR[1]=realYPR[1];}
 }
}
```

2.4 Analýza autonomního řízení

Vzhledem k nárokům automatického řízení si nevystačím se současným osazením kvadrokoptéry, protože můj gyroskop obsahuje pouze měření náklonu a zrychlení, což nestačí pro aplikaci autonomního řízení. Potřebujeme ještě navíc informaci o úhlovém natočení kvadrokoptéry vzhledem k nějakým pevně umístěným vztažným bodům na zemském povrchu. Vztažným bodem dle kterého se lze orientovat může být cokoliv, ale je kontraproduktivní nějaké určovat, když se mi nabízí velmi jednoduché řešení v podobě zemských pólů, respektive rovník a nultý poledník. Jednoduše mi k určení polohy vůči nějakému výchozímu bodu poslouží běžně dostupný GPS[3] modul se zabudovaným magnetometrem. Magnetometr není podmínkou, ale velmi podstatným způsobem zjednoduší práci – musel bych nejprve přes složitější matematické výpočty nakalibrovat natočení vůči zemským pólům pomocí „pokus/omyl“, kdy bych poslal kvadrokoptéru malou rychlostí náhodným směrem a následně podle změny přijímaných souřadnic z GPS modulu dopočítal natočení vůči zemským pólům.

Tím pádem pokud bych měl informaci o směru natočení čela kvadrokoptéry a zeměpisnou polohu, už lehce bych kvadrokoptéru navedl daným směrem a pouze bych kontroloval zda-li se už aktuální souřadnice polohy nerovnají souřadnicím požadovaným. Problém nastává se signálem z družic, který je lehce „zastaven“ (oslaben). Proto GPS přijímače téměř nefungují v budovách či v blízkosti budov a tzn. že při

[3]Stručný popis funkce GPS viz kapitola 2.4.2

aplikaci autonomního řízení by bylo nutné zařadit mechanismy na zastavení letu při ztrátě signálu a schopnost vrátit se do místa, ve kterém byl ještě signál dostatečně silný. Tzn. Arduino Due by muselo zapisovat do paměti nejlépe celou trasu letu, aby pak dokázalo navádět let zpět, případně do místa, kde ještě autonomní let probíhal bez potíží.

2.4.1 Příklad možnosti realizace autonomního řízení

Zadání souřadnic, kam má kvadrokoptéra autonomně doletět, by probíhalo např. přes wi-fi, např. přes internetový prohlížeč. Zde bych také mohl sledovat pohyb kvadrokoptéry na mapě v reálném čase. Jako GPS modul by se dal použít Ublox NEO-6M GPS Module propojený s Arduinem přes UART nebo SPI sběrnici. Ještě bych připojil tří-osý magnetometr HMC5883L přes TWI. Příklad vyčítání hodnot ze zmiňovaného GPS modulu při použití UART:

```
1 static void smartDelay(unsigned long time)
2 {
3  unsigned long start_time=millis();
4  do {
5   while (Serial1.available()) {gps.encode(Serial1.read());}
6  } while((millis()-start_time) < time);
7 }
```

2.4.2 GPS – Global Positioning System

Global Positioning System, zkráceně GPS, globální družicový polohový systém provozovaný Ministerstvem obrany USA. S pomocí GPS systému družic lze určit jednoznačné zeměpisné souřadnice (sférické souřadnice) na jakémkoli místě na Zemi:

- Stupně severní/jižní šířky – úhlová vzdálenost od rovníku
- Stupně východní/západní délky – úhlová vzdálenost od 0. poledníku (Greenwich ve Velké Británii)

Přesnost GPS se udává do 10 m (pro běžné neautorizované uživatele – turistika, doprava atd.), ale lze ji zvýšit až na jednotky centimetrů (pro autorizované uživatele (vojenský sektor USA a vybrané spojenecké armády za účelem navádění střel, přesné mapování bojiště apod.). Používá se pasivní přijímač, který přijímá signály z jednotlivých viditelných družic (jsou v danou chvíli nad obzorem). Z přijatých dat vypočítá polohu antény, nadmořskou výšku a zobrazí informaci o přesném času. Současně na střední oběžné dráze obíhá 24 družic ve výšce 20350 km nad povrchem Země na 6 kruhových drahách se sklonem 55° a rychlostí 3,8 km/s s dobou oběhu kolem Země 11 h 58 min – polovina siderického dne. Medián množství viditelných družic je v ČR 8, s minimem 6 a maximem 12 družic. Pro určení polohy jsou nutné nejméně 3 družice (pouze ale 2D zobrazení, tzn. bez nadmořské výšky), ale s větším počtem družic, ze

kterých přijímač dostává informace pro zpracování roste i přesnost. U 4 přijímaných družic se nám umožní i výpočet nadmořské výšky.

Určování polohy pomocí GPS je jednoduše řečeno založeno na zasílání tzv. časových razítek každou družicí – čas, kdy byl datagram odeslán ze satelitu – a dále informace o poleze své a dalších satelitů. V přijímači se vypočítá rozdíl mezi časem přijmutí a odeslání datagramu. Problém nastává v tom, že družice obsahují atomové hodiny s přesností až desetiny pikosekundy, které jsou ale finančně náročné na pořízení, takže nemůžou být umístěny i v přijímačích pro standardní aplikace (turistika apod.). Proto se musí pomocí složitých algoritmů výchozí čas korigovat a pomocí dat ze družice synchronizovat. Následně pokud dojde k vypočtení doby po kterou signál putoval od družice k přijímači, tak na základě znalosti rychlosti šíření signálu je již snadné dopočítat jak daleko se družice od přijímače nachází. A nakonec ze znalosti vzdálenosti přijímače od družice a polohy samotné družice je založeno celé měření. Nutné podotknout, že na přesnosti času závisí celé měření a jelikož je rychlost šíření signálu zhruba stejná jako rychlost šíření světla, jsou zde velké nároky na přesnost na úrovni nano až piko sekund. Např. chyba měření času, kdy přijímač pochybí jen o 1 μs, je následná odchylka polohy 300m.

2.5 Možnosti telemetrie

Pro přenos telemetrie se nabízí hned několik způsobů:

- Pomocí šestého kanálu mého přijímače Hott GR-12L
- Wi-Fi shield pro Arduino DUE pro přenos do PC/mobilního telefonu
- Bluetooth shield pro Arduino pro přenos do PC/mobilního telefonu
- GSM (GPRS) modul pro komunikaci s mobilním telefonem

Přenos šestým kanálem přijímače Hott GR-12L

První možností je tedy použití 6. kanálu mého přijímače Hott GR-12L, který je určen pro přenos telemetrie na úrovni určitého modelářského standardu. Problém je, že tento přijímač je určen pro více modelových řad vysílačů a můj vysílač je z řady, která nepodporuje příjem telemetrie. Tzn. musel by být pořízen odpovídající přijímač. Z hlediska programátorského, kdy při použití kteréhokoli jiného z výše uvedených způsobů si lze programově ošetřit velikost paketu, přenášené informace apod., jelikož by informace neprocházeli přes Arduino. Dále bych byl limitován příjmem na malý diplej nového přijímače a omezeným počtem přenášených údajů – informace se omezují zejména na stav ESC a jejich nastavení, případně kapacitu baterie apod. (tzn. základní provozní informace). Následně bych i musel pořídit komunikační jednotku mezi přijímačem a senzory a nebo zvolit složitější řešení – zjistit

způsob komunikace a komunikaci vést přes Arduino. Takže přenos pomocí 6. kanálu je praktický pro modeláře pro základní provozní informace.

Wi-Fi shield

Při použití wi-fi shieldu pro Arduino se mi nabízí rozsáhlá možnost přenosu telemetrie a to v podstatě všeho co budu chtít přenášet. Stačí pouze příslušný senzor připojit k Arduinu Due způsobem, jakým lze číst jeho hodnoty a následně zpracovat a poslat pomocí wi-fi do přijímače, kterým může být např. PC, mobilní telefon apod. Připojení na wi-fi shield z přijímací stanice je velmi jednoduché, kdy do internetového prohlížeče pouze napíšeme IP adresu wi-fi shieldu a zobrazí se mi vysílané informace. Případně bych si pak mohl pro větší přehlednost udělat např. grafické rozhraní, kdy by se mi otevřela další možnost řízení kvadrokoptéry pomocí wi-fi, kdy bych mohl zadávat přes prohlížeč souřadnice pro autonomní let, nebo jen měnit letové parametry, jako rychlost letu, výkon motorů apod. a v podstatě za chodu zasahovat do běžícího programu (samozřejmě na úrovni obsahu proměnných a ne měnit v průběhu části kódu).

Výhodou tedy je velmi jednoduchý způsob komunikaci, její uskutečnění a velký rozsah možných úkonů. Zejména stremování videa přímo z letu díky možnému velkému datovému přenosu. Ale to by jsem nesměl posílat video přes Arduino, jelikož by video datový tok rapidně vytěžoval Arduino. Naopak nevýhodou je nedostačující dosah pohybující se kolem 30 m ve volném prostoru (vzduchu).

Bluetooth shield

V případě požití blutooth shieldu, získáme zhruba stejné možnosti telemetrie a řízení kvadrokoptéry na dálku, jako u wi-fi shieldu. Ovšem při nižší přenosové rychlosti. Rozdíl je v tom, že bluetooth má ještě omezenější dosah a průchodnost překážkami než wi-fi. Pak z hlediska ovládání kvadrokoptéry a čtení přijímaných informací, mi již nestačí webový prohlížeč, ale potřebuji program na vysílání a přijímání informací od kvadrokoptéry.

GSM modul

GSM modul by po-té byl schopen uskutečnit hovor nebo zaslat SMS zprávu či poslat omezenou rychlostí data přes připojení k internetu pomocí GPRS. Tím získám efektivní možnost na dálku řídit kvadrokoptéru a možnost zasílat telemetrické údaje pomocí SMS zpráv nebo přes internet. A to i na velkou vzdálenost. Ovšem s omezenou rychlostí GPRS, takže nepřipadá v úvahu video, ale pouze maximálně zasílat fotky s omezeným rozlišením.

ZÁVĚR

Fyzicky byl sestrojen prototyp pomocí modelářských komponent. Následně jsem zjistil jak tyto komponenty ovládat, či přijímat z nich informace. ESC regulátory (či jinak frekvenční měniče) jsem nejprve ovládal PPM modulací, ale později jsem zjistil, že vlastním novější typ ESC, takže je můžu ovládat i PWM modulací o frekvenci 490 Hz. Takže jsem všechny komponenty propojil způsobem odpovídajícím jejich správnému fungování a přikročil ke psaní zdrojového kódu. Rozkol mezi napětím, na kterém funguje Arduino Due (3,3 V) a ostatní použité periferie (5 V), jsem jednoduše ošetřil pomocí *IOREF* pinu (viz kap. 1.2), kde jsem přivedl referenční napětí 5 V, takže na výstupech Due je nyní 5 V místo 3,3 V. Analogicky jsem po této operaci mohl přivádět na vstupy Due též 5 V.

Na začátku psaní kódu jsem hledal způsoby, jak komunikovat s okolím z Arduina Due z programátorského hlediska. Ať už se jedná o přijímání signálů nebo tvorba PWM na výstupech. Ohledně přijímání signálů bylo řešení jednoznačné – použití standardní Arduino funkce `pulseIn`, kde jsem ale teprve po delších době pokusů konečně zjistil, že je nutné zakázat přerušení pro její správnou funkci. Větší problém nastal při generování PWM a PPM. Standardní Arduino funkce zde nefungovali správně, protože Arduino Due je jako jedno z mála Arduin postaveno na ARM architektuře (32 bitové), načež ostatní jsou založeny na AVR architektuře. A zároveň Arduina Due není v Arduino komunitě příliš populární. Po určité době jsem se dostal ke knihovně Motor32, která měla být napsaná právě proto, aby řešila výše zmíněný rozkol mezi ARM a AVR architekturou (zejména problém s časovači, jak jsem i ověřil na osciloskopu). S jejím použitím generovalo přesně co jsem požadoval – fázově správnou 490 Hz PWM a PPM modulaci s periodou 50 ms. Tímto byl vyřešen i problém jak řídit motory. Po-té tedy šlo jen si zhruba navrhnout algoritmus (viz obr.5 2.1) a ten následně přepsat v jazyku, kterému Arduina Due rozumí (viz kap. 2.3).

Povedlo se napsat řídící kód, který je laboratorně otestován. Řídící výstupy z Arduina Due i silové výstupy z ESC jsem sledoval na osciloskopu a podrobil zátěžovým testům a vše vyšlo dle očekávání – vše se generovalo dle mých požadavků. Po-té jsem tedy zkoušel najít regulační konstanty pro PID, aby se povedlo kvadrokoptéru stabilizovat (prototyp držea ve výšce za jednu osu, zatímco jsem druhou zkoušel). Došel jsem tedy k závěru, že návrh odpovídajícího PID je mimo rámec mých možností. Ale i s mým PID se povedlo prototyp částečně stabilizovat, což bylo velmi časově náročné, jelikož jsem musel regulační konstanty zjišťovat způsobem pokus-omyl. Lze tedy říci, že bez odpovídajícího PID nelze kvadrokoptéru lépe stabilizovat. Nepomohla ani následná redukce prodlevy obdržení aktuální hodnoty náklonu z DMP (zrychlení TWI, vyřešení FIFO fronty).

Na konec jsem stručně zanalyzoval možnosti autonomního letu pomocí GPS modulu a nastínil způsob zapracování řízení v kódu. A po-té promyslel pár možností pro přenos telemtrie a zhruba zhodnotil jejich realizaci, výhody a nevýhody. Bluetooth vyšlo jako téměř nepoužitelné pro přenos telemetrie a řízení kvadrokoptéry díky malému dosahu. Naopak wi-fi se hodí pro rozsáhlý přenos dat v obou směrech (streamování videa), ale použití je limitováno malým dosahem v řádu desítek metrů. Jako střední cesta vyšlo použití GSM, respektive GPRS modulu. Získáme velký dosah na úkor malého datového přenosu, který je ale dostatečný pro přenos stručných zpráv obsahující telemetrické informace. Nebo zasílání pokynů směrem ke kvadrokoptéře.

LITERATURA

[1] *Teknlife* [online]. [cit. 27. 11. 2015]. Dostupné z URL: <http://cdn.teknlife.com/wp-content/uploads/2014/06/drone1200.jpg>.

[2] Arduino Due. *Arduino.cc* [online]. [cit. 2. 11. 2015]. Dostupné z URL: <https://www.arduino.cc/en/Main/ArduinoBoardDue>.

[3] Compare board specs. *Arduino.cc* [online]. [cit. 2. 11. 2015]. Dostupné z URL: <https://www.arduino.cc/en/Products/Compare>.

[4] Cortex-M3 Processor. *Arm.com* [online]. [cit. 3. 11. 2015]. Dostupné z URL: <http://www.arm.com/products/processors/cortex-m/cortex-m3.php>.

[5] Cortex-M3 – základní informace. *Ucsimply.cz* [online]. [cit. 3. 11. 2015]. Dostupné z URL: <http://www.ucsimply.cz/cm3/popis-procesoru/zakladni-informace/>.

[6] HRBÁČEK, Jiří. *Komunikace mikrokontroléru s okolím.* 1. vyd. Praha: BEN - technická literatura, 2000, 151 s. ISBN 80-860-5673-2.

[7] FRÝZA, Tomáš, Zbyněk FEDRA a Jiří ŠEBESTA. *Mikroprocesorová technika: laboratorní cvičení.* Vyd. 1. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2008, 50 s. ISBN 978-80-214-3756-2. Dostupné také z URL: <http://www.urel.feec.vutbr.cz/BMPT/media/bmpt_laboratore.pdf>.

[8] Pulzně šířková modulace PWM: Teoretické poznatky – Obrázek 4.1: Princip tvorby PWM signálu. FRÝZA, Tomáš, Zbyněk FEDRA a Jiří ŠEBESTA. *Mikroprocesorová technika: laboratorní cvičení.* Vyd. 1. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2008, s. 1. ISBN 978-80-214-3756-2. Dostupné také z URL: <http://www.urel.feec.vutbr.cz/BMPT/media/bmpt_laboratore.pdf>.

[9] ŠIŠKA, Martin. *Impulsové modulace* [online]. 2013 [cit. 10. 11. 2015]. Dostupné z URL: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=68045>. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Radim Číž Ph.D.

[10] *opentx.cz* [online]. [cit. 10. 11. 2015]. Dostupné z URL: <http://www.opentx.cz/images/9/98/RC_Receiver_Timing_Diagram.jpg>.

[11] MPU-6050 Accelerometer + Gyro. *Arduino.playground.cc* [online]. [cit. 15. 11. 2015]. Dostupné z URL: <http://playground.arduino.cc/Main/MPU-6050>.

[12] MPU-6000 and MPU-6050 Product Specification Revision 3.4. *Invensense.com* [online]. 2013 [cit. 11. 12. 2015]. Dostupné z URL: <http://43zrtwysvxb2gf29r5o0athu.wpengine.netdna-cdn.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>.

[13] BLÁHA, Martin. *Elektronicky komutovaný motor* [online]. 2008 [cit. 20. 11. 2015]. Dostupné z URL: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=8615>. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Petr Melichar.

[14] *avdweb.nl* [online]. [cit. 20. 11. 2015]. Dostupné z URL: <http://www.avdweb.nl/Article_files/Solarbike/Motor-controller/4-Pole-brushless-DC-motor-animation.jpg>.

[15] EMAX Simon Series 20A For Muti-Copter. *Emaxmodel.com* [online]. [cit. 25. 11. 2015]. Dostupné z URL: <http://www.emaxmodel.com/multicopter/simonk-series-esc/emax-simon-series-20a-for-muti-copter.html>.

[16] *aliexpress.com* [online]. [cit. 20. 11. 2015]. Dostupné z URL: <http://g03.a.alicdn.com/kf/HTB17GKqIXXXXXcwXVXXq6xXFXXXg/Emax-Simonk-Series-12A-20A-30A-ESC-For-Quadcopter-QAV250-.jpg>.

[17] VRBA, Marek. *Nabíječka lithiových akumulátorů* [online]. 2012 [cit. 5. 12. 2015]. Dostupné z URL: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=59212.Diplomovápráce.VysokéučenítechnickévBrně.VedoucípráceJuanBautistaArroyoGarcia.>.

[18] 3EB4030 – FOXY G2 – LC Li-Pol 4500mAh/11,1V 40/80C 50,0Wh. *tatramodel.sk* [online]. [cit. 5. 12. 2015]. Dostupné z URL: <http://www.tatramodel.sk/zakladna-ponuka/akumulatory-a-baterie/akumulatory-li-pol/foxy/3EB4030-foxy-g2-lc-li-pol-4500mah111v-4080c-500wh/?>.

[19] Mz-10, 5 channel HoTT radio control DE. *Graupner.de* [online]. [cit. 7. 12. 2015]. Dostupné z URL: <http://www.graupner.de/en/products/1736df13-32af-4183-aa8e-80f31a7f03cb/S1001/product.aspx>.

[20] MZ-10 2,4GHz HOTT RC souprava. *heureka.cz* [online]. [cit. 7. 12. 2015]. Dostupné z URL: <https://im9.cz/iR/importprodukt-orig/3af/3afafed732c5f34f4a4a4a3d6c5844cf.jpg>.

[21] Aalborg University–Department of electronic systems. *Es.aau.dk* [online]. [cit. 7. 12. 2015]. Dostupné z URL: <http://kom.aau.dk/~petarp/papers/DAFH-AFR.pdf>.

[22] Ultrasonic Ranging Module HC-SR04. *micropik.com* [online]. [cit. 6. 12. 2015]. Dostupné z URL: <http://www.micropik.com/PDF/HCSR04.pdf>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

2D	2-Dimension – dvourozměrný prostor/zobrazení
3D	3-Dimension – třírozměrný prostor/zobrazení
A	Amper – SI jednotka velikosti elektrického proudu
AC	Alternate Current – střídavý proud (napětí)
ACK	ACKnowledge – potvrzení přijetí určitého množství dat
AD	Analogově-Digitální – převodník/převod z analogového signálu na digitální
Ah	Ampérhodina – jednotka kapacity baterie
ARM	Advanced RISC Machine – zdokonalená architektura RISC mikroprocesoru
B	Bajt – 8 bitů
Bd	Baud – jednotka modulační rychlosti, také jako symbolová nebo znaková rychlost
b/bit	Bit – binární bit
BEC	Battery Eliminator Circuit – stabilizační obvod v ESC
BLDC	BrushLess DC electric motor – bezkomutátorový DC motor
CPU	Central Processing Unit – centrální procesorová jednotka
DA	Digitálě-Analogový – převodník/převod z digitálního signálu na analogový
DC	Direct Current – stejnosměrný proud (napětí)
DMP	Digital Motion Processor
EMF	ElectroMotive Force – elektromotorická síla
FHSS	Frequency Hopping Spread Spectrum – metoda přenosu v rozprostřeném spektru
FH-CDMA	Frequency-Hopping Code Division Multiple Access – metoda mnohonásobného přístupu k zařízení

G	Giga
g	Gram – jednotka hmotnosti
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications, originálně z francouzského Groupe Spécial Mobile
H	High – vysoká hodnota ve dvoustavové logice
h	Hodina – jednotka času
Hz	Hertz – množství cyklických dějů odehraných za jednu sekundu
I^2C	Inter-Integrated Circuit – multi-masterová sériová sběrnice
I/O	Input/Output – vstup/výstup
IP	Internet Protocol
JTAG	Joint Test Action Group – standard pro testování plošných spojů, programování flash pamětí
k	Kilo
KV	Charakteristika elektromotoru – počet otáček na jeden volt
L	Low – nízká hodnota ve dvoustavové logice
Li-Ion	Lithium-Iont – lithium-ionové akumulátory
Li-Pol	Lithium-Polymer – lithium-polymerový akumulátor
LSB	Least Significant Bit – řádově nejméně významný bit
M	Mega
m	Mili/Metr
min	Minuta – jednotka času
mm^2	Milimetr čtvereční
MSB	Most Significant Bit – řádově nejvýznamnější bit
PID	Proporcionální, Integrační a Derivační regulátor

PPM	Pulse Position Modulation – pulsní polohová modulace
PWM	Pulse Width Modulation – pulsně šířková modulace
RC	Remote Control – dálkově ovládané
s	Sekunda – základní SI jednotka času
SCL	Synchronous CLock – vodič pro posílání hodinového signálu v synchronní sériové sběrnici
SDA	Synchronous DAta – datový vodič v synchronní, sériové sběrnici
SPI	Serial Peripheral Interface – sériové periferní rozhraní, typ sběrnice
SRAM	Static Random Access Memory – statická RAM paměť
TWI	Two Wire Interface – (dvouvodičové rozhraní) prakticky identická sběrnice jako I^2C, pouze realizována mj. na mikrokontrolérech Atmel
UART	Universal Asynchronous Receiver and Transmitter – universální asynchronní rozhraní
USB	Universal Serial Bus – universální sériová sběrnice
USB OTG	USB On-The-Go – standard umožňující zařízením komunikovat jako hostitelské zařízení pro USB přístroje
V	Volt – jednotka rozdílu elektronických potenciálů
Wi-Fi	Wireless Fidelity
μ	Mikro

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

- BP_DMajer_164761.pdf – elektronická verze bakalářské práce ve formátu pdf
- BP_final_code.zip – zip soubor obsahující zdrojový kód a použité knihovny
- drone_BP.jpg – foto sestrojeného prototypu